Complex Fuzzy Logic Reasoning-based Methodologies for Quantitative Software Requirements Specifications

Dan E. Tamir[1], Carl J. Mueller[2], and Abraham Kandel[3]

[1]Department of Computer Science, Texas State University, San Marcos, Texas, USA

[2]Department of Computer Information Systems, Texas A&M University - Central Texas, Killeen, Texas, USA

[3]School of Computing and Information Sciences, Florida International University, Miami, Florida, USA

Abstract

Quantitative software engineering is one of the most important paradigms for software development. That is, Requirements, Analysis, Design, Coding, and Testing. One of the challenges associated with quantitative software engineering is the fact that many of the quantifiable parameters are concomitant with uncertainty. Part of the uncertainty is because a significant portion of the software engineering process involves human beings presenting rational, yet difficult to quantify, behavior. Due to this fact, soft computing approaches, specifically fuzzy logic based reasoning, present significant opportunities for constructing sound quantitative software engineering models.

This work presents a new and innovative approach for fuzzy logic based quantitative software engineering procedures. We present a complex fuzzy logic based inference system used to account for the intricate relations between software engineering constraints such as quality, software features, and development effort. The new model concentrates on the requirements specifications part of the software engineering process. Moreover, the new model significantly improves the expressive power and inference capability of the soft computing component in the soft computing based quantitative software engineering.

Key Words: Fuzzy logic – Fuzzy set theory – Fuzzy Inference - Complex fuzzy logic – Complex fuzzy set theory – Complex fuzzy Inference – Software engineering – Software development – Software requirements specifications - Quantitative software engineering – Quantitative software development – Quantitative software requirements specifications - Computational intelligence

1 Introduction

Since Software Engineering’s introduction in 1968, one of the challenges facing practitioners is to eliminate the uncertainties arising from the chaotic nature of software development [1]. One of the most widely known outcomes of the 1st International Conference on Software Engineering in Garmisch, Germany was the detection of the existence of a gap between the available design and implementation practices and the complexity of the software under development [1]. The gap and crisis identified in Garmisch relates to the notion that the tools and techniques used to develop computer software are inadequate for the complexity of the needed software. Consequently, much of the software engineering research conducted following this conference focuses on providing tools and techniques for reducing the uncertainty and assuring the quality of software with ever-growing complexity. Of a specific concern is the uncertainty related to the requirements specifications phase. As the first phase in the development chain, the requirement specification phase has a profound effect on the quality of the entire software development process and on the final product, i.e., the hardware/software system.

One of the approaches for reducing uncertainty in the software development process, adopted by mainstream software engineering researchers, has been to introduce the discipline of quantitative software engineering. The quantitative software engineering research and practice stream, however, cannot completely cope with uncertainty, as some of this uncertainty is inherent to the process. Moreover, the fact that software engineering is a human-intensive process adds a challenging uncertainty dimension since human beings' reasoning is often characterized by inexact and fuzzy logic. This prompted a new interdisciplinary collaborative research direction that combines knowledge from the disciplines of uncertainty management and mitigation and the field of software engineering. Numerous research efforts in the area have been conducted, and many papers addressing soft computing and quantitative software engineering have been published [2, 3, 4, 5]. Fuzzy logic is one of the most commonly and successfully used "tools" for handling uncertainty [6, 7, 8, 9, 10]. Indeed some papers addressing the role of fuzzy logic in quantitative software engineering have been published [11, 12, 13, 14].

This chapter presents a new and innovative approach for fuzzy logic based quantitative software engineering procedures. The proposed complex fuzzy logic based model enables reasoning about processes with multi-dimensional components where each component is carrying fuzzy information and the interaction between the components cannot be decomposed and represented via primitive, one dimensional, fuzzy set theory and fuzzy logic operations such as conjunction, disjunction, negation, union, and intersection. In specific, we present the foundations of a complex fuzzy logic based inference system used to account for the intricate relations between software engineering constraints such as quality, software features, and development effort. The new model concentrates on the requirements specifications part of the software engineering process. Our model significantly improves the expressive power and inference capability of classic fuzzy logic as the tool for handling the uncertainty in this environment.

The problem addressed in this chapter boils down to the suitability of fuzzy logic as a soft computing model for dealing with uncertainty in software requirements specifications in tandem with applying quantitative software engineering methods. We ascertain that the fuzzy logic approach is a strong and excellent methodology for handling the uncertainty that is inherent in quantitative software engineering. Nevertheless, we show that the traditional single dimension fuzzy logic might fall short on dealing with real-world problems where several features such as quality, cost, development time, and usability, are involved. Especially, when these features are intertwined in a way that cannot be reduced to traditional fuzzy logic expressions composed of basic fuzzy logic connectives (conjunction, disjunction, negation, etc.).

The solution proposed is to use complex fuzzy logic as the underlying theory for dealing with the uncertainty involved in software requirements specifications. Via constructive examples we show that complex fuzzy logic is highly suitable for the task at hand.

The main contribution of the research described in the chapter is the formulation of a model that can enable better handling of the uncertainty in quantitative software engineering. To the best of our knowledge, this is the first research that is exploring the utility of complex fuzzy logic for handling uncertainty in the framework of quantitative software engineering. Furthermore, the research, which concentrates on software requirements specifications, can be extended to other phases of the software development process.

The rest of the chapter is organized in the following way. Section 2 and Section 3, respectively, provide background concerning uncertainty involved in the software development process and the Quality Function Deployment approach to software requirements specifications. Section 4 contains a literature review listing relevant work. Section 5 introduces the concept of complex fuzzy logic and presents some ways it can be used for inference in the context of uncertainty in quantitative software requirements specifications. Finally, Section 6 includes the conclusions.

2 Uncertainty and the Software Development Process

One of the first techniques addressing the uncertainty and the growing complexity in the software development process was the Waterfall software development model introduced at the IEEE WESCON conference in 1970 by Winston Royce [15]. In this seminal paper on the software development process; Royce introduced the first formalization of the software development process; organizing it into a series of five major processes of: Requirements, Analysis, Design, Coding, and Testing. Over the years, researchers have made a number of changes to the model. One of the most significant changes was the absorption of the Analysis phase into the Requirements and Design phases. With the addition of minor name changes, the Waterfall development model has evolved to include four phases that many associate with the model: Requirements, Design, Implementation or Construction, and Validation.

Even though the Waterfall model has served the software development industry well for almost 50 years, there are a number of problems with the model. One of the most significant issues is that it focuses on reducing machine utilization with a result of increased personnel utilization, thereby making software development a very labor-intensive process. Another issue with the Waterfall model is that the software developers are dependent on the quality of the requirement specifications established in the first phase of the process when analysts and developers know the least about the application. Frequently, these requirements originate from sources that do not understand the information necessary to build software and have a limited knowledge of the application, causing the developers to have questions about the requirement specifications. One of the major causes of costly software maintenance or project failure is poor requirement specifications [16, 17]. As developers began to gain more experience with the Waterfall model, they started to investigate a number of techniques to resolve the amount of human labor necessary to produce high quality software products and to reduce the impact of vague or incomplete software requirement specifications.

One of the first proposed approaches to address the uncertainty of software requirements was prototyping. Software prototypes can have two forms: throwaway and evolutionary [18]. A throwaway prototype provides information about the general structure and layout of the software but does not provide any information about the operation. A major disadvantage of this approach is that at the end of the design phase developers discard the prototype. Although this approach provides a great deal of information about user interfaces and links, it is expensive; and generally, it is not popular with the financial stakeholders within an organization. As its name implies, an evolutionary prototype is a working model of the desired software implemented without the use of traditional quality control tools. An evolutionary prototype becomes version-0, and the test engineers have the task of assuring that there are no defects in the software. From this version-0, analysts reverse engineer the software to create any required documentation.

Because of the many issues with prototyping, software engineers have turned to the notion of iterative software development. Iterative software development is a maintenance-based strategy used to reduce both risk and uncertainty during the construction of the application. One of the most widely known iterative techniques is Barry Boehm's Spiral Model [19]. In addition to the Spiral Model, most of the agile development methods also employ this concept for the same reasons [20, 21, 22].

In addition to the risks and uncertainty that are inherent in developing software, there is a great deal of uncertainty in describing the features needed in the software. Extracting the user needs and describing these needs in a format understandable by non-technical and technical individuals provides a source of considerable uncertainty in software engineering. Two of the major sources of uncertainty in the process of establishing the specifications for the needed software are the software engineers and the non-technical individuals providing the information upon which to base these specifications. Since it is unlikely that researchers will resolve all of the challenges in human communications any time soon, it is probably better to defer this challenge for future research. One of the tools being proposed as a first step in addressing these communication challenges is changing the perspective of requirement specifications to focusing on the tasks that the software's operators perform. A second tool proposed is utilizing soft computing methodologies for handling this uncertainty. Fuzzy logic has been successfully applied to resolve uncertainty at each of the five major processes of the waterfall model [23, 24, 25, 26]. In this chapter, however, the soft computing model proposed is complex fuzzy logic.

3 Requirements Specification via Quality Function Deployment

In the 1970's, requirements engineers began to formulate a notion of the information that is necessary to develop a software application. These efforts evolved into the development of the IEEE Recommended Practice for Software Requirements Specifications, which is divided into sections describing the required interface, software functionality, non-functional or quality requirements, and constraints [16, 17]. The IEEE Recommended Practice-model centers on the items that are necessary for the software engineers to build the software. Although this general model has served the software industry very well, it does not provide a view of the software from the end user's perspective. The IEEE recommendation views security and usability as quality issues and documents them as non-functional requirements. This may explain the reason that these areas have remained challenges for software developers. Documenting software requirements from the perspective of the end-user or software operator (a.k.a. human centric) is an approach that is gaining in popularity. A human-centric approach to eliciting and documenting software requirements concentrates on the tasks that the software must support and who performs those tasks [27, 28, 29]. Generalizing this notion of viewing software from an operator's perspective yields the concepts of viewing software requirements from the external tasks (performed by operators and/or machines) that the software is intended to support. The Unified Modeling Language (UML) implements this concept in its use case diagram [30]. Software modeling techniques are also evolving to support the change to a user-centric approach [30, 31, 32]. One of the changes to software modeling techniques is the practice of employing use cases or user stories to describe the high-level characteristics of an application.

All of these innovations have acted to reduce the uncertainty of defining specifications and developing software applications, but there are many areas where the opportunity to further reduce challenges in software development activity exists. One such area is selecting the order of implementing the software requirements. Selecting the order of implementing requirements can permit an early deployment of the product or service.

A number of innovative techniques have migrated into modern software development practice from research conducted by Japanese investigators in the 1960's and 1970's into improving manufacturing and quality assurance. One of the techniques making the migration from quality assurance into software development is Quality Function Deployment (QFD). Shigeru Mizuno and Yoji Akao conducted research directed at bringing quality assurance into the design phase, rather than in the manufacturing phase resulting in QFD [33]. Their vision was to include the customers' view of the quality into all aspects of product design and manufacturing, thereby increasing the acceptance of the product in the marketplace [34].

According to Richard Zultner, applying QFD to software requirements is a relatively simple process [35]. Customers receive a copy of the specification for a specific application; they then assign one of three QFD categories to each requirement. These classification categories are *normal*, *expected*, and *exciting*. A customer classifying a requirement as normal means that a product like the one specified **has** that feature. A requirement classified as expected means that the customer believes a product not containing that feature is disappointing. A requirement receiving an exciting classification is one exceeding what the customer expects to find in the specified product. There are two challenges not addressed in Zultner's discussion on QFD: Customer priority assignment and using QFD throughout the development process [35].

After receiving the customers' individual classification of the requirements, requirements engineers have several methods for establishing the classification of each requirement. One of these methods of applying customer priorities is to assign the requirement classification receiving the most votes. Another approach is to record the votes for each category providing later processes with more data for decision-making.

Although simple, QFD presents a number of challenges to requirements engineers. One challenge that requirements engineers are facing is selecting the customers for providing the classifications because the quality of data is dependent on the customers' knowledge of the product and/or market. Another challenge for QFD relates to the quality of the customers asked to classify requirements. A QFD classification does not provide the requirements engineer insight into missing requirements. An advantage of QFD, outweighing both of these challenges, is that it provides the requirements engineer with customer insight as to the value of the specified facilities.

A challenge related to the customer's skill in evaluating requirements, but one that requirements engineers can control, is the focus and structure of the requirements. There are three major sections in a traditional requirement specification: interface, functional, and non-functional requirements [16, 17]. An issue arising with this type of document is that the interface, function and performance specifications are in three different locations making it difficult to pull all of this information together and classify each of the features. An approach that can improve requirement classification accuracy is an external-task or user-centric specification [28, 30, 31]. A user-centric specification differs from traditional specification in that the requirements are organized based on the task that a customer is intended to perform with the software. Expressing the software's functionality in terms of use cases or user stories based on tasks they will perform with the software will produce better classifications [30, 31].

One of the possible uses, later in the development process, for these priorities is establishing implementation priorities. Prioritizing feature implementation is a significant challenge facing software developers using iterative development techniques such as Berry Boehem's Spiral model or Ken Schwalbe's Agile Scrum [19, 21]. In both the spiral model and Agile Scrum, developers must select a set of features for implementation during the next iteration. Usually, developers accomplish this by selecting features, based on effort estimates, fitting to the duration of the development increment. A better selection approach employs both the QFD classification and estimated effort. Using these two factors is even more appropriate for situations where one or more iterations will result in phase deployment or release.

Based on the definition of the QFD categories, it is apparent that it is an ordinal scale where requirements in an expected category are more desirable than requirements in the normal category; and requirements in the exciting category are more desirable than requirements in the expected category. Using this scale for development priorities would mean that exciting requirements are developed first followed by expected and then normal; but to have each iteration possess the maximum desirability to the customer base, the development priorities are expected requirements, followed by normal requirements, and then exciting. Implementing expected requirements is critical because they are the requirements that can increase customer dissatisfaction with the deployed product. Exciting requirements can increase the marketability of the product but might not improve customer satisfaction. Therefore, implementing the normal requirements before implementing the exciting requirements increases customer satisfaction and assures that the product is equal to the completion.

Designing the development process to work only on the requirements in a specific category is not a guarantee that developers will produce software maximizing the development time and overall customer satisfaction. Because of the complexity of the variables, traditional algorithmic approaches are not viable. What is needed is a new approach to produce a list of requirements in rank order for an iteration cycle.

The discussion in Section 2, has presented the software development process and the uncertainty involved in the process. The current section (3) concentrated on using quantifiable methodologies for software specifications. It is quite clear that the quantification process reduces yet does not eliminate uncertainty. When it is all said and done, the engineers and stakeholders have to make decisions that optimize a utility, effort, and risk function. This function, however, is “ill defined” due to the inherent uncertainty and the fuzzy nature of human communication and human reasoning. For example, assigning ranks such as normal, expected, and exciting is a classic example of (human) fuzzy logic based reasoning. In this chapter, we propose to formulize two of the dimensions of the QFD space, namely utility and effort, using complex fuzzy logic. Later on, risk can be added as a third dimension in a multi-dimensional complex fuzzy logic based QFD process. In the next section, we list relevant work.

4 Literature Review

This section includes a review of literature associated with software requirements and describes work related to the use of fuzzy logic in formulating methods for handling uncertainty in software development. A new and innovative method for handling the uncertainty, which is proposed in this chapter, is the utilization of complex fuzzy logic. This original method is further elaborated in the next section.

A review of recent literature for software requirements reveals a limited amount of recent investigation into ways for writing and organizing requirements. Books like Karl Wiegers’ *Software Requirements*, Soren Lauesen’s *Software Requirements: Style and Techniques*, and Dean Leffingwell’s *Agile Software Requirements* discuss most of the research into writing and organizing requirements [36, 37, 38]. Each of these texts investigates most of the core issues of requirements analysis, but they do not investigate using formal methods for dealing with the uncertainty inherent in the process. Although similar, each text presents the topic from differing perspectives: traditional, linguistic, and lean software development methodologies, such as Agile-Scrum.

In the book *Software Requirements,* Karl Weigers investigates most of the issues relating to the development of traditional requirements specification documents and the management of those requirements throughout the development process [36]. One of the features that make this book an important resource for the topic of software requirements is that it provides a large number of examples on eliciting requirements in a business environment. Even though Weigers addresses almost every aspect of software requirements, some might argue that the areas of specification style and Agile requirements practice need additional investigation. In the chapter addressing writing software requirements, Wiegers provides an excellent discussion on the mechanics of writing specifications, but he does not discuss the effects of different styles. In the book *Software Requirements: Styles and Techniques*, the author provides a better discussion on this issue. On lean software development or Agile methodologies [37], the discussion explains some of the differences between traditional requirements elicitation and the approach introduced with Agile-Scrum, but does not address the way that these differences affect the developers and the stakeholders. In the book *Agile Software Requirements*, Leffingwell provides a view of the effects of requirements on the developers and stakeholders [38].

The book *Software Requirements: Styles and Techniques* by Soren Lauesen provides an overview to the requirements elicitation process, but focuses on linguistics techniques for achieving a specific objective [37]. Like Karl Wiegers’ approach, Lauesen provides a large number of cases studies and examples in writing requirements to achieve specific results and illustrates that different writing styles can achieve different results. This work, however, does not address ways for writing and organizing the requirements in order to enable software development using an Agile development methodology.

One of the most unusual approaches to software requirements specifications is described in the book *Agile Software Requirements* by Dean Leffingwell [38]. In this work, Leffingwell combines Agile Modeling with requirements analysis and describes the ways that requirements are used in Agile development methodologies. The book suggests that requirements have a hierarchical characteristic, which is a subtle change from the “flat” approach suggested in other works. Using a hierarchical approach provides a level of details that is appropriate to the stakeholder and the developer.

One of the deficiencies that almost all of the works on software requirements have in common is their way of treatment of non-functional requirements, a.k.a. Quality Requirements or "ileitis". Originally, non-functional requirements were addressing system level topics such as reliability and maintainability. Over time, other topics such as human factors and security were introduced under non-functional requirements because many experts viewed these topics as system level issues that did not directly relate to the functionality of the software. Today two of the most severe challenges to software engineers are software usability and security.

In recent years, there has been a significant interest in the area of quantitative software engineering [2, 3, 4, 5]. Several papers have addressed computational intelligence and quantitative software engineering [11, 12, 13, 14]. Additionally, several survey papers and books/ book-chapters such as [39, 40, 41, 42, 43] are useful in gaining access into recent developments in the field.

Alongside the interest in the general area of computational intelligence and software engineering, there has been increasing interest in the use of fuzzy set theory and fuzzy logic based reasoning as the soft computing paradigm [44, 45, 46, 47, 48, 49]. With this respect [44, 45] are some of the most comprehensive accounts on fuzzy logic models in quantitative software engineering. The utilization of fuzzy logic to quantitative software engineering makes a lot of sense and provides highly valuable and usable tools for coping with the uncertainty in quantitative software engineering [44, 45, 46, 47, 48, 49]. Nevertheless, this approach falls short of providing a rich and expressive way to take into account the intricate relations between major parameters affecting the software development process, such as quality, usability, development effort, and features included in release, cost, reliability, and risk. It is our assertion that the intricate relations can be effectively addressed using complex fuzzy logic.

Complex fuzzy logic has been introduced by Ramot et al. [50, 51] and several related applications have been considered [52]. Tamir et al. refined the definition provided by Ramot and introduced examples where the interpretation provides for a rich and effective paradigm for reasoning which can capture uncertainty and human reasoning in a highly effective way [53, 54, 55, 56].

An exhaustive search in research databases did not reveal any work that connects complex fuzzy logic with quantitative software engineering. To the best of our knowledge, this is the first research effort that reports on such a research direction.

5 Complex Fuzzy Systems

Several aspects of the software requirements specifications can utilize the concept of complex fuzzy logic [53]. Complex fuzzy logic can be used to represent the two-dimensional information embedded in the description of tradeoffs between design effort and software feature inclusion. Additionally, complex fuzzy logic based inference can be utilized to exploit the fact that variables related to the uncertainty are inherent in the software requirements specifications. The software requirements space is multi-dimensional and cannot be readily defined via single dimensional clauses connected by single dimensional connectives. Finally, the multi-dimensional fuzzy space defined as a generalization of complex fuzzy logic can serve as a media for clustering of specifications related information in a linguistic variable-based feature space.

Tamir et al. introduced a new interpretation of complex fuzzy membership grade and derived the concept of pure complex fuzzy classes [53, 55]. This section introduces the concept of a pure complex fuzzy grade of membership, the interpretation of this concept as the denotation of a fuzzy class, and the basic operations on fuzzy classes.

To distinguish between classes, sets, and elements of a set we use the following notation: a class is denoted by an upper case Greek letter, a set is denoted by an upper case Latin letter, and a member of a set is denoted by a lower case Latin letter.

The Cartesian representation of the pure complex grade of membership is given in the following way:

$\mu(V,z) = \mu_r(V) + j\mu_i(z)$,

where $\mu_r(V)$ and $\mu_i(z)$, the real and imaginary components of the pure complex fuzzy grade of membership, are real value fuzzy grades of membership. That is, $\mu_r(V)$ and $\mu_i(z)$ can get any value in the interval [0,1]. The polar representation of the pure complex grade of membership is given by:

$\mu(V,x) = r(V)e^{j\sigma\phi(z)}$,

where $r(V)$ and $\phi(z)$, the amplitude and phase components of the pure complex fuzzy grade of membership, are real value fuzzy grades of membership. That is, they can get any value in the interval [0,1]. The scaling factor σ is in the interval $(0,2\pi)$. It is used to control the behavior of the phase within the unit circle according to the specific application. Typical values of σ are $\{1, \frac{\pi}{2}, \pi, 2\pi\}$. Without loss of generality, for the rest of the discussion in this section we assume that $\sigma = 2\pi$.

The difference between pure complex fuzzy grades of membership and the complex fuzzy grade of membership proposed by Ramot et al. [50, 51] is that both components of the membership grade are fuzzy functions that convey information about a fuzzy set. This entails a different interpretation of the concept as well as a different set of operations and a different set of results obtained when these operations are applied to pure complex grades of membership. This is detailed in the following sections.

5.1 Complex Fuzzy Class

A fuzzy class is a finite or infinite collection of objects and fuzzy sets that can be defined in an unambiguous way and comply with the axioms of fuzzy sets given by Tamir et al. and the axioms of fuzzy classes given by [53, 54, 57, 58]. While a general fuzzy class can contain individual objects as well as fuzzy sets, a *pure fuzzy class of order one* can contain only fuzzy sets. In other words, individual objects cannot be members of a pure fuzzy class of Order 1. A pure fuzzy class of order M is a collection of pure fuzzy classes of order $M-1$. We define a *Complex Fuzzy Class* Γ to be a pure fuzzy class of order one, i.e., a fuzzy set of fuzzy sets. That is, $\Gamma = \{V_i\}_{i=1}^{\infty}$; or $\Gamma = \{V_i\}_{i=1}^{N}$ where V_i is a fuzzy set and N is a finite integer. Note that despite the fact that we use the notation $\Gamma = \{V_i\}_{i=1}^{\infty}$, we do not imply that the set of sets $\{V_i\}$ is enumerable. The set of sets $\{V_i\}$ can be finite, countably infinite, or uncountably infinite. The use of the notation $\{V_i\}_{i=1}^{\infty}$ is just for convenience.

The class Γ is defined over a universe of discourse T. It is characterized by a pure complex membership function $\mu_\Gamma(V,z)$ that assigns a complex-valued grade of membership in Γ to any element $z \in U$ (where U is the universe of discourse). The values that $\mu_\Gamma(V,z)$ can receive lie within the unit square or the unit circle in the complex plane and are in one of the following forms:

$\mu_\Gamma(V,z) = \mu_r(V) + j\mu_i(z)$,

$\mu_\Gamma(z,V) = \mu_r(z) + j\mu_i(V)$,

where $\mu_r(\alpha)$ and $\mu_i(\alpha)$, are real functions with a range of [0,1].

Alternatively:

$\mu_\Gamma(V,z) = r(V)e^{j\theta\phi(z)}$,

$\mu_\Gamma(z,V) = r(z)e^{j\theta\phi(V)}$,

where $r(\alpha)$ and $\phi(\alpha)$, are real functions with a range of [0, 1] and $\theta \in (0,2\pi]$.

In order to provide a concrete example, we define the following pure fuzzy class. Let the universe of discourse be the set of all the features that can be added to a specific software application along with a set of attributes related to the features, such as related development effort and perception of importance (i.e., expected, normal, exciting). Let M_i denote the set of features considered in step i of the software development process. Furthermore, consider a function (f_1) that associates a number between 0 and 1 with each set of features, where this function reflects the level of importance of the features included in the set. In addition, consider a second function (f_2) that associates a number between 0 and 1 with each specific feature, where this function denotes the development effort associated with including the feature in step i of the software development process. The functions (f_1, f_2) can be used to define a pure fuzzy class of order 1. A compound of the two functions in the form of a complex number can represent the degree of membership in the pure fuzzy class of "highly desired features" for the set of features considered in the last k development steps.

Formally, let U be a universe of discourse and let 2^U be the powerset of U. Let f_1 be a function from 2^U to [0, 1] and let f_2 be a function that maps elements of U to the interval [0, 1]. For $V \in 2^U$ and $z \in U$ define $\mu_\Gamma(V,z)$ to be:

$$\mu_\Gamma(V,z) = \mu_r(V) + j\mu_i(z) = f_1(V) + jf_2(z)$$

Then, $\mu_\Gamma(V,z)$ defines a pure fuzzy class of order 1, where for every $V \in 2^U$, and for every $z \in U$, $\mu_\Gamma(V,z)$ is the degree of membership of z in V and the degree of membership of V in Γ. Hence, a complex fuzzy class Γ can be represented as the set of ordered triples: $\Gamma = \{V, z, \mu_\Gamma(V,z) | V \in 2^U, z \in U\}$

Depending on the form of $\mu_\Gamma(\alpha)$ (Cartesian or polar), $\mu_r(\alpha)$, ì$_i(\alpha)$, $r(\alpha)$, and $\phi(\alpha)$ denote the degree of membership of z in V and/or the degree of membership of V in Γ. Without loss of generality, however, we assume that $\mu_r(\alpha)$ and $r(\alpha)$ denote the degree of membership of V in Γ for the Cartesian and the polar representations respectively. In addition, we assume that $\mu_i(\alpha)$ and $\phi(\alpha)$ denote the degree of membership of z in V for the Cartesian and the polar representations respectively. Throughout this chapter, the term *complex fuzzy class* refers to a pure fuzzy class with pure complex-valued membership function, while the term *fuzzy class* refers to a traditional fuzzy class such as the one defined by [57].

5.2 Degree of Membership of Order *N*

The traditional fuzzy grade of membership is a scalar defining a fuzzy set. It can be considered as degree of membership of order 1. The pure complex degree of membership defined in this chapter is a complex number that defines a pure fuzzy class. That is, a fuzzy set of fuzzy sets. This degree of membership can be considered as degree of membership of order 2 and the class defined can be considered as a pure fuzzy class of order 1. Additionally, one can consider the definition of a fuzzy set (a class of order 0) as a mapping into a one-dimensional space and the definition of a pure fuzzy class (a class of order 1) as a mapping into a two-dimensional space. Hence, it is possible to consider a degree of membership of order N as well as a mapping into an N-dimensional space. The following is a recursive definition of a fuzzy class of order N. Part 2 of the definition is not necessary; it is given in order to connect the term pure complex fuzzy grade of membership and the term grade of membership of order 2.

Definition:

1) A fuzzy class of order 0 is a fuzzy set; it is characterized by a degree of membership of order 1 and a mapping into a one-dimensional space.
2) A fuzzy class of order 1 is a set of fuzzy sets. It is characterized by a pure complex degree of membership. Alternatively, it can be characterized by a degree of membership of order 2 and a mapping into a two-dimensional space.
3) A fuzzy class of order N is a fuzzy set of fuzzy classes of order N-1; it is characterized by a degree of membership of order $N + 1$ and a mapping into an $(N + 1)$-dimensional space.

5.3 Generalized Complex Fuzzy Logic

A general form of a complex fuzzy proposition is: "x ... A ... B ..." where A and B are values assigned to linguistic variables and " ... " denotes natural language constants. A complex fuzzy proposition P can get any pair of truth values from the Cartesian interval $[0,1] \times [0,1]$ or the unit circle. Formally a fuzzy interpretation of a complex fuzzy proposition P is an assignment of fuzzy truth value of the form $p_r + jp_i$, or of the form $r(p)e^{j\theta(p)}$, to P. In this case, assuming a proposition of the form "x ... A ... B ...," then p_r ($r(p)$) is assigned to the term A and p_i ($\theta(p)$) is assigned to the term B.
For example, under one interpretation, the complex fuzzy truth value associated with the complex proposition:

"x is an expected yet highly difficult to implement feature of the application"

can be $0.1 + j0.5$. Alternatively, in another context, the same proposition can be interpreted as having the complex truth value $0.3e^{j0.2}$. As in the case of traditional propositional fuzzy logic, we use the tight relation between complex fuzzy classes / complex fuzzy membership to determine the interpretation of connectives. For example, let C denote the complex fuzzy set of "features that are exciting and easy to implement," and let $f_C = c_r + jc_i$, be a specific fuzzy membership function of C, then f_C can be used as the basis for the interpretation of P. Next we define several connectives along with their interpretation.

Table 1 includes a specific definition of connectives along with their interpretation. In this table, P, Q, and S denote complex fuzzy propositions and $f(S)$ denotes the complex fuzzy interpretation of S. We use the fuzzy Łukasiewicz logical system as the basis for the definitions [57, 59]. Hence, the max t-norm is used for conjunction and the min t-conorm is used for disjunction. Nevertheless, other logical systems, such as Gödel fuzzy systems, can be used [59, 60].

Table 1. Basic Propositional Fuzzy Logic Connectives

Operation	Interpretation
Negation	$f('P) = 1 + j1 - f(P)$
Disjunction	$f(P \oplus Q) = \min(p_R, q_R) + j \times \min(p_I, q_I)$
Conjunction	$f(P \otimes Q) = \min(p_R, q_R) + j \times \min(p_I, q_I)$
Implication	$f(P \longrightarrow Q) = \min(1, 1 - p_R + q_R) + j \times \min(1, 1 - p_I + q_I)$

The same axioms used for fuzzy logic are used for complex fuzzy logic, and Modus ponens is the rule of inference.

5.4 Complex Fuzzy Propositions and Connectives Examples

Consider the following propositions(P, Q, and S respectively):

P: "x is a very **exciting** yet highly **difficult to implement** feature. "

Q: "x is **expected** yet quite **easy to implemen**t feature. "

S: "x is a high **ranked** feature planned for release in the near **future**. "

Let A be the term *"x is an exciting feature,"* and let B denote the term *"difficult to implement."* Furthermore, let C be the term *"x is an expected feature,"* let D be the term *"x is a high ranked feature,"* and let E be the term *"future."* Hence, P is of the form: "x is a very A that is highly B," and Q is of the form "x is C that is not quite B." In this case, the terms

'"$expected,$" "$normal,$" "$difficult,$" "$ranked,$", and "$future$" are linguistic variables. Furthermore, a term such as "$exciting,$" can get fuzzy truth values (between 0 and 1) or fuzzy linguistic values such as "$moderately$," "$highly,$" and "$very,$" (the terms "$is,$" "$that,$" etc. are linguistic constants). Assume that the complex fuzzy interpretation (i.e., degree of confidence or complex fuzzy truth value) of P is $p_r + jp_i$, while the complex fuzzy interpretation of Q is $q_r + jq_i$. Thus, the truth value of "$x\ is\ an\ exciting\ feature,$" is p_R, and the truth value assigned to "$x\ is\ difficult\ to\ implement,$" is p_i. The truth value of "$x\ is\ an\ expected\ feature,$" is q_r. Suppose that the term "$easy$" stands for "$not\ \boldsymbol{difficult},$" the term "$low,$" stands for "$not\ high,$" and the term "$dull$" stands for "$not\ \boldsymbol{exciting}$". In a similar way, S is of the form: "$x\ is\ high\ D\ that\ is\ \ldots\ near\ E,$" where the complex fuzzy interpretation of S is $s_r + js_i$. This, however, is not the only way to define these linguistic terms, and it is used to exemplify the expressive power and the inference power of the logic. Hence, the complex fuzzy interpretation of the following composite proposition is:

1) $f('p) = (1 - p_r) + j(1 - p_I)$

That is, $'P$ denotes the proposition "x is a dull yet easy to implement feature."

The confidence level in $'P$ is $(1 - p_r) + j(1 - p_i)$, where the fuzzy truth value of the term "x is a non **exciting** feature," is $(1 - p_r)$ and the fuzzy truth value of the term "x is an **easily implemented** feature." is $(1 - p_i)$

2) $f(P \oplus Q) = \max(p_r, 1 - q_r) + j \times \max(p_i, 1 - q_i)$.

That is, $(P \oplus Q)$ denotes the proposition "x is an **exciting** yet highly **difficult to implement** feature." OR

"x is an **expected** yet quite **easy to implemen**t feature." The truth values of individual terms, as well as the truth value of $P \oplus {}'Q$ are calculated according to Table 1.

3) $f('P \otimes Q) = \min(1 - p_r, q_r) + j \times \min(1 - p_i, q_i)$.

That is, $('P \otimes Q)$ denotes the proposition "x is a **dull** yet **difficult to implement** feature." AND

"x is an **expected** yet quite **easy to implemen**t feature." The truth values of individual terms, as well as the truth value of $'P \otimes Q$ are calculated according to Table 1.

4) Let the term R stand for $(P \oplus Q)$, (the complex fuzzy interpretation of R is $r_r + jr_i$.) then, $\mathrm{R} \longrightarrow \mathrm{S} = \min(1, 1 - r_r + s_r) + j \times \min(1, 1 - r_i + s_i)$

Thus, $(\mathrm{R} \longrightarrow \mathrm{S})$denotes the proposition

IF "x is an **exciting** yet **difficult to implement** feature." OR

"x is an **expected**, yet quite **easy to implemen**t feature."

THEN "$x\ is$ a high **ranked** feature planned for release in the near **future**." The truth values of individual terms, as well as the truth value of $\mathrm{R} \longrightarrow \mathrm{S}$ are calculated according to Table 1.

5.5 Complex Fuzzy Inference Example

Assume that the degree of confidence in the proposition R as defined above is $r_r + jr_i$, and assume that the degree of confidence in the fuzzy implication $T = R \longrightarrow S$ is $t_r + jt_i$. Then, using Modus ponens

R

$\underline{R \longrightarrow S}$

S

one can infer S with a degree of confidence $\min(r_r, t_r) + j \times \min(r_i, t_i)$.

In other words if one is using:

"x is an **exciting** yet **difficult to implement** feature. " OR

"x is an **expected** yet **easy to implemen**t feature. "

IF "x is an **exciting** yet **difficult to implement** feature. " OR

"x is an **expected** yet **easy to implemen**t feature. "

THEN "x is a high **ranked** feature planned for release in the near **future**. "

"x is a high **ranked** feature planed for release in the near **future**. "

Hence, using Modus ponens one can infer:

"x is a high **ranked** feature planned for release in the near **future**" with a degree of confidence of $\min(r_r, t_r) + j \times \min(r_i, t_i)$.

This example shows the potential of complex fuzzy inference to enhance the ability for resolving uncertainty involving the requirements specifications process. The actual process of using this approach for inference is described [51]. In this case a complex fuzzy rule-based system is generated via complex fuzzification and used for complex fuzzy inference. Eventually via de-fuzzification actual crisp conclusions are obtained [51]. In [26] we have described Software Testing Using Artificial Neural Networks and Info-Fuzzy Networks. We are currently working on extending this research to using complex fuzzy inference. Finally, we are currently exploring the use of complex fuzzy logic and inference for non-functional requirements such as usability requirements.

6 Conclusions

In this chapter, we have introduced an innovative approach for fuzzy logic based quantitative software engineering procedures. We have presented a complex fuzzy logic based inference system used to account for the intricate relations between software engineering constraints such as quality, software features, and development effort. The model presented concentrates on the requirements specifications part of the software engineering process. Furthermore, the presented model significantly improves the expressive power and inference capability of the soft computing component in the soft computing based quantitative software engineering.

In the future, we plan to concentrate on software requirements for human computer interaction applications. Additionally, we plan to further investigate the utility of the new model in the development of software requirements for large-scale software systems. Furthermore, we plan to increase the dimensionality of the fuzzy terms to include other factors such as risk, reliability, usability etc. Finally, we plan to expand the work to include other components of the software development process.

7 References

[1] "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968," Scientific Affairs Division, NATO, Brussels, 1969.

[2] M. Kim, T. Zimmermann and N. Nagappan, "An Empirical Study of Refactoring Challenges and Benefits at Microsoft," *Software Engineering, IEEE Transactions on,* vol. 40, no. 7, pp. 633-649, 2014.

[3] M. Lazaro and E. Marcos, "An Approach to the Integration of Qualitative and Quantitative Research Methods In Software Engineering Research," in *CAISE*06 Workshop on Philosophical Foundations on Information Systems Engineering*, Luxemburg, 2006.

[4] T. Dyba, "An empirical investigation of the key factors for success in software process improvement," *Software Engineering, IEEE Transactions on ,* vol. 31, no. 5, pp. 410-424, 2005.

[5] J. M. Verner and W. M. Evanco, "In-house software development: what project management practices lead to success?," *IEEE Software,* vol. 22, no. 1, pp. 338-353, 2005.

[6] L. A. Zadeh, "Fuzzy sets," *Information and Control,* no. 8, pp. 338-353, 1965.

[7] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems,* no. 1, pp. 3-28, 1978.

[8] A. Kandel, Fuzzy Mathematical Techniques with Applications, Reading, MA: Addison-Wesley, 1986.

[9] D. E. Tamir and A. Kandel, "An axiomatic approach to fuzzy set theory," *Information Sciences,* no. 52, pp. 75-83, 1990.

[10] D. E. Tamir and A. Kandel, "Fuzzy semantic analysis and formal specification of conceptual knowledge," *Information Sciences, Intelligent systems,* no. 82(3-4), pp. 181-196, 1995.

[11] S. K. Pillai and M. K. Jeyakumar, "Evaluation of neural networks for software development effort estimation using a new criterion," *SigSoft Software Engineering Notes,* vol. 39, no. 5, pp. 1-6, 2014.

[12] G. Nagendra Kumar and C. Aswani Kumar, "Generation of high level views in reverse engineering using formal concept analysis," in *First International Conference on Networks & Soft Computing (ICNSC)*, Hyderabad, 2014.

[13] C. Singh, A. Pratap and A. Singhal, "Estimation of software reusability for component-based systems using soft computing techniques," in *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference -*, Noida, 2014.

[14] T. Bakshi, B. Sarkar and S. K. Sanyal, "A new soft-computing based framework for project management using game theory," in *Communications, Devices and Intelligent Systems (CODIS), 2012 International Conference on* , Kolkata, 2012.

[15] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *IEEE WESON 26 (August): 1-9*, Los Angeles, 1970.

[16] IEEE, "IEEE Std 1233-1998 IEEE Guide for Developing System Requirement Specification," IEEE Computer Society, New York, NY, 1998.

[17] IEEE, "IEEE Std-830-1988 IEEE Recommended Practice for Software Requirements," IEEE Computer Society, New York, NY, 1998.

[18] R. Pressman, in *Software Engineering: A Practioner's Approach*, New York, NY, McGraw-Hill, 2010.

[19] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer,* vol. 21, no. 5, pp. 61-72, 1988.

[20] S. W. Ambler, "Agile Modeling," Ambysoft, Inc., 2014. [Online]. Available: http://www.agilemodeling.com. [Accessed 30 9 2014].

[21] K. Schwaber and M. Beedle, Agile Software Development with Scrum, Upper Saddle River, NJ: Prentice-Hall, 2002.

[22] K. Beck and C. Andres, Extreme Programming Explained: Embrace Change 2nd ed, Addion-Wesley Professional, 2004.

[23] W. Pedrycz, G. Succi, M. Reformat, P. Musilek and X. Bai, "Self-organizing Maps as a Tool for Software Analysis,," in *Canadian Conference of Electrical and Computer Engineering 2001*, Toronto, Canada, May 2001.

[24] J. Noppen, P. van den Broek and M. Aksit, "Dealing with fuzzy information in software design methods," in *Proceedings of the IEEE Annual Meeting of the North American Fuzzy Information Processing NAFIPS '04*, Alberta, Canada, June 2004.

[25] P. Achimugu, A. Selamat, R. Ibrahim and M. Mahrin, "An Adaptive Fuzzy Decision Matrix Model for Software Requirements Prioritization," in *Advanced Approaches to Intelligent Information and Database Systems*, vol. 551, J. B. V. C. S. Sobecki, Ed., Springer International Publishing, 2014, pp. 129-138.

[26] D. Agrawal, D. E. Tamir, M. Last and A. Kandel, "A Comparative Study of Software Testing Using Artificial Neural Networks and Info-Fuzzy Networks," *IEEE, Transactions on Man Machine and Cybernetics,* vol. 42, no. 5, pp. 1183-1193, February 2012.

[27] A. M. Hickey and A. M. Davis, "A Unified Model of Requirements Elicitation," *Journal of Management Information Systems,* vol. 20, no. 4, pp. 65-84, 2004.

[28] L. L. Constantine and L. A. D. Lockwood, Software for Use: A practical Guide to the models and methods of Usage-Centered Design., Reading, MA: ACM Press, 1999.

[29] A. Sutcliffe, "Scenario-based Requirments Analysis," *Requirements Engineering,* vol. 3, no. 1, pp. 48-65, 1998.

[30] G. Booch, J. Rumbaugh and I. Jacobon, Unified Modeling Language User Guide, 2nd Ed., New York, NY: Addison-Wesley Professional, 2005.

[31] S. W. Ambler, "Agile Modeling: Effective Practices of Modeling and Documentation," Ambysoft Inc., 2014. [Online]. Available: http://www.agilemodeling.com/. [Accessed 11 November 2014].

[32] I. Jacobson, Object Oriented Software Engineering: A Use Case Driven Approach, Redwood City, CA USA: Addison Wesley Longman Publishing Co., 2004.

[33] G. Mazur, "History of QFD," Quality Function Deployment Institute, [Online]. Available: http://ww.gfdi.org/what_is_qfd/history_of_qfd.html. [Accessed 29 September 2014].

[34] Y. Akao, Quality Function Deployment: Integrating Customer Requirements into Product Design, Cambridge, MA: Productivity Press, 1990.

[35] R. E. Zultner, "Quality Function Deployment for Software: Satisfying Customers," *American Programmer,* pp. 28-41, Febrary 1992.

[36] K. Wiegers and J. Beatty, Software Requirements, Readmond, WA: Microsoft Press, 2013.

[37] S. Lauesen, Software Requirements: Styles and Techniques, Edinburg Gate: Pearson Education, 2002.

[38] D. Leffingwell, Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise, Boston, MA: Pearson Education, Inc., 2011.

[39] M. K. Bhuyan, D. P. Mohapatra and S. Sethi, "A survey of computational intelligence approaches for software reliability prediction," *ACM SIGSOFT Software Engineering Notes,* vol. 39, no. 2, pp. 1-10, 2014.

[40] M. Harman, S. A. Mansouri and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Survey,* vol. 45, no. 1, p. 61, 2012.

[41] J. Lee, Software Engineering with Computational Intelligence, Secaucus, NJ: Springer-Verlag, 2003.

[42] S. Dick and A. Kandel, Computational Intelligence in Software Quality Assurance, Singapore: Workd Scientific, 2005.

[43] M. Last and A. Kandel, "Automated Test Reduction Using an Info-Fuzzy Network," in *Software Engineering with Computational Intelligence*, vol. 731, Springer International, 2003, pp. 235-258.

[44] W. Pedrycz and G. Succi, "Fuzzy Logic Classifiers and Models in Quantitative Software Engineering," in *Advances in Machine Learning Applications in Software Engineering*, IGI Global, 2007, pp. 146-

167.

[45] W. Pedrycz, A. Breuer and N. J. Pizzi, "Fuzzy Adaptive Logic Networks as Hybrid Models of Quantitative Software Engineering," *Intelligent Automation and Soft Computing,* vol. 12, no. 2, pp. 189-209, 2008.

[46] J. Lee and J.-Y. Kuo, "New approach to requirements trade-off analysis for complex systems," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 10, no. 4, pp. 551-562, 1998.

[47] O. Georgieva and A. Dimov, "Software reliability assessment via fuzzy logic model," in *Proceedings of the 12th International Conference on Computer Systems and Technologies (CompSysTech '11)*, Vienna, Austria, 2011.

[48] K. Cooper, J. W. Cangussu, R. Lin, G. Sankaranarayanan, R. Soundararadjane and E. Wong, "An Empirical study on the specification and selection of components using fuzzy logic," in *Proceedings of the 8th international conference on Component-Based Software Engineering (CBSE'05)*, St. Louis, 2005.

[49] R. George, R. Srikanth, F. E. Petry and B. P. Buckles, "Uncertainty management issues in the object-oriented data model," *Fuzzy Systems, IEEE Transactions on,* vol. 4, no. 2, pp. 179-192, 1996.

[50] D. Ramot, R. Milo, M. Friedman and A. Kandel, "Complex Fuzzy Sets," *Fuzzy Systems, IEEE Transactions on,* vol. 10, no. 2, pp. 171-186, 2002.

[51] D. Ramot, M. Friedman, G. Langholz and A. Kandel, "Complex Fuzzy Logic," *Fuzzy Systems, IEEE Transactions on,* vol. 11, no. 4, pp. 450-461, 2003.

[52] S. Dick, "Towards complex fuzzy logic," *Fuzzy Systems, IEEE Transactions on,* vol. 13, no. 3, pp. 405-414, 2005.

[53] D. E. Tamir, J. Lin and A. Kandel, "A New Interpretation of Complex Membership Grade," *International Journal of Intelligent Systems,* vol. 26, no. 4, 2011.

[54] D. E. Tamir and A. Kandel, "Axiomatic Theory of Complex Fuzzy Logic and Complex Fuzzy Classes," *International Journal of Computers, Communications & Control,* vol. 6, no. 3, 2011.

[55] D. E. Tamir, M. Last and A. Kandel, "Complex Fuzzy Logic," in *On Fuzziness*, R. Seising, E. Trillas, S. Termini and C. Moraga, Eds., Springer, 2013, p. 429.

[56] D. E. Tamir, M. Last and A. Kandel, "The Theory and Applications of Generalized Complex Fuzzy Propositional Logic," in *Soft Computing: State of the Art Theory and Novel Applications*, R. R. Yager, A. M. Abbasov, M. Z. Reformat and S. N. Shahbazova, Eds., Springer, 2013.

[57] L. Behounek and P. Cintula, "Fuzzy class theory," *Fuzzy Sets and Systems,* vol. 154, no. 1, pp. 34-55, 2005.

[58] A. A. Fraenkel, Y. Bar-Hillel and A. Levy, Foundations of Set Theory, 2nd ed., Amsterdam: Elsivier, 1973.

[59] P. Cintula, "Advances in LΠ and LΠ1/2 logics," *Archives of Mathematical Logic,* vol. 42, pp. 449-468, 2003.

[60] F. Montagna, "On the predicate logics of continuous t-norm BL-algebras," *Archives of Mathematical Logic,* vol. 44, pp. 97-114, 2005.