01 Apr 1998

Fuzzy Requirements

Xiaoqing Frank Liu
Missouri University of Science and Technology, fliu@mst.edu

Recommended Citation

X. F. Liu, "Fuzzy Requirements," *IEEE Potentials*, Institute of Electrical and Electronics Engineers (IEEE), Apr 1998.
The definitive version is available at https://doi.org/10.1109/45.666642

Fuzzy requirements

Trying to make everyone feel good about the product goals specified

Xiaoqing Frank Liu

Requirement analysis and specification is the first major step in software development. The goal is to develop a requirements' specification that contains all the customers' true needs. The analysis describes quality requirements and their constraints, such as cost and resources. Functional requirements are analyzed in terms of inputs, outputs and their relationships.

Hence, requirements analysis enables software engineers to specify software function and establish software design and implementation constraints. Frederick P. Brooks, Jr. has pointed out that no other part of the work so cripples the resulting system if done wrong. What's more, no other part is harder to rectify later. Defects in the requirements specification can be propagated and amplified in the design and the coding phase. However, many challenges hinder the wide application of requirement engineering methodologies and techniques in software development organizations.

Requirements are sometimes not specified and documented in detail in many software development projects. This makes software validation and software maintenance very difficult. One challenge is that many product requirements are fuzzy in nature. In fact, customers usually describe their requirements in fuzzy terms such as *good, high, low, very important*. Translating these general terms into design specifications that will accurately create the desired product is difficult. Another challenge is that requirements often conflict with each other, especially those from different perspectives. For example, the efficiency requirement from customers often conflicts with the maintainability requirement from maintenance personnel. Moreover, many conflicts between requirements are implicit and difficult to identify. For example, a conflict between the usability and the efficiency requirements is sometimes hard to determine.

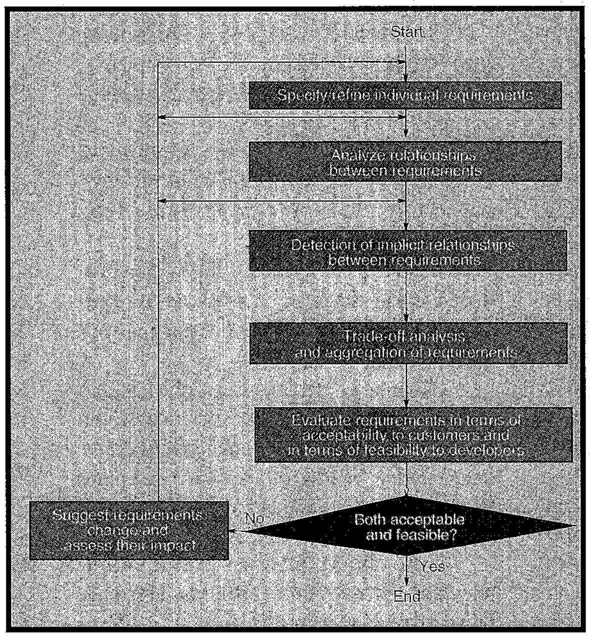

Start
Specify/refine individual requirements
Analyze relationships between requirements
Detection of implicit relationships between requirements
Trade-off analysis and aggregation of requirements
Evaluate requirements in terms of acceptability to customers and in terms of feasibility to developers
Both acceptable and feasible?
No
Suggest requirements change and assess their impact
Yes
End

Fig. 1 A requirement engineering process for fuzzy requirement analysis

A requirement engineering process

There are two important goals in requirement engineering: 1) to acquire requirements that are satisfactory to their customers, and 2) to generate feasible requirements. These two objectives often compete with each other. A requirement that satisfies customers to the highest degree is often not practical. Conversely, the requirement most feasible from the system developer's viewpoint usually falls short in meeting customer expectations.

To achieve both objectives, the requirements often need to be refined many times. The refinement of fuzzy requirements with complex relationships between them is an iterative process. Negotiations among conflicting requirements can be very difficult and time-consuming. A formal trade-off analysis can help. A requirement engineering process for analysis is shown in Fig. 1.

Individual requirements from different perspectives are first acquired and represented formally. The relationships between them are then analyzed. Implicit conflicts can be detected based on fuzzy inference techniques. Multiple requirements are collated to formulate an overall system requirement based on fuzzy, multi-criteria, decision making techniques; in which, an objective is often formulated by combining multiple criteria using fuzzy operators, such as fuzzy conjunctives, in fuzzy logic.

Defining the individual and the overall system requirements permits verifying and validating that product requirements have been met. If requirements are evaluated to be unsatisfactory to customers or infeasible to developers, they need to be reformulated. If thought to be unacceptable to the customer, they should be strengthened while maintaining their feasibility. If a problem for the developer, they should be relaxed while maintaining customer satisfaction as much as possible. The resulting overall impact should be assessed fully before requirements are relaxed or strengthened.

Specifying individual fuzzy requirements using fuzzy sets

The most effort is placed on representing system requirements. Fuzzy requirements often represent elastic criteria against which the acceptability of a realized target system is judged. Some impose constraints on the system's development process; others describe desired properties, such as cost. An example would be:

R_c: The software development cost should be *LOW.*

Some fuzzy requirements impose constraints on the final system; others describe desired system features. An example would be:

R_l: The software system should be *EASY* to learn.

The constraint imposed by fuzzy requirement *R* can be represented as a satisfaction function, denoted as Sat_R. This maps an element of *R*'s domain *D* to a number in [0,1], which represents how well the requirement is satisfied:

Sat_R: D → [0,1].

In essence, the satisfaction function characterizes a fuzzy subset of *D* that satisfies the fuzzy requirement. For example, the elasticity of the requirement R_l can be captured using the satisfaction function. This corresponds to the membership function of the fuzzy set *EASY* in the requirement (see Fig. 2). On the graph, on the vertical axis, 1 represents that the requirement is completely satisfied, while 0 means not meeting the requirement at all.

Classifying relationships

There are four types of significant relationships between requirements: 1) conflicting, 2) cooperative, 3) mutually exclusive and 4) irrelevant. The classification is determined by how satisfying one requirement impacts on the satisfaction degree of another requirement.

Two requirements are *conflicting* if raising satisfaction in one requirement "often" decreases the other's level of satisfaction. If it "always" decreases the satisfaction degree of the other, they are said to be *completely conflicting.*

On the other hand, two requirements are *cooperative* if increasing satsifaction in one often increases the degree the other is satisfied. If the rise in satisfaction of one always increases satisfaction in the other, they are *completely cooperative.*

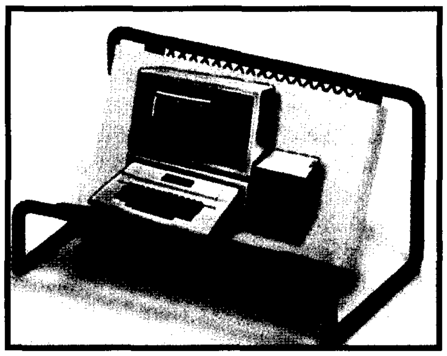
So how do we characterize partially conflicting and cooperative requirements? A conflicting degree $0 \leq conf(R_i,R_j) \leq 1$ or cooperative degree $0 \leq coop(R_i,R_j) \leq 1$ can be attached to a relationship between requirement R_i and R_j. If $conf(R_i, R_j)$ is 1, R_i completely conflicts with R_j. If $conf(R_i, R_j)$ is 0, they do not conflict with each other at all. If $coop(R_i, R_j)$ is 1, R_i is completely cooperative with R_j. If $coop(R_i, R_j)$ is 0, they are not cooperative at all. Thus,

1976—Steve Jobs and Steve Wozniak found the Apple Computer Company on April Fool's Day. Apple Computer introduces the Apple I system.

1980—Apple Computer becomes a publicly held company, selling 4.6 million shares at $22 (US) per share. More than 40 Apple employees and investors become instant millionaires.

1982—Apple Computer becomes the first personal computer company to reach $1 billion (US) in annual sales.

1984—Apple Computer runs its "1984" commercial during the SuperBowl introducing the Macintosh computer.



© Apple Computer, Inc

one can define terms such as "highly conflicting" or "somewhat conflicting" using membership functions in Fig. 3.

Sometimes two requirements cannot be satisfied all at the same time. That is, to satisfy one fuzzy requirement somewhat, the other cannot be satisfied at all, and vice versa. We refer to them as *mutually exclusive requirements.*

Of course, two requirements may be irrelevant. Satisfying one requirement to any degree does not have any impact on the degree another requirement is satisfied.

Detecting implicit relationships

Many conflicts are implicit and difficult to identify, especially in large scale software systems. Therefore, it helps to have techniques to identify implicit conflicting and cooperative relationships between requirements. Several heuristics can be used to infer relationships between requirements based on the identified relationships.

• *Heuristic rule 1* (infer relationships from cooperative requirements):

Let D be either a domain shared by three requirements R1, R2 and R3. If requirement R1 is cooperative with R2 in D, R2 is cooperative with R3 in D, and they are not irrelevant, then R is cooperative with R3 in D.

• *Heuristic rule 2* (infer relationships from conflicting and cooperative requirements):

Let D be either a domain shared by three requirements R1, R2 and R3. If requirement R1 is cooperative with R2 in D, R2 conflicts with R3 in D, and they are not irrelevant, then R1 is conflicting with R3 in D.

• *Heuristic rule 3* (infer relationships from conflicting requirements):

Let D be either a domain shared by three requirements R1, R2 and R3. If requirement R1 conflicts with R2 in D, R2 conflicts with R3 in D, and they are not irrelevant, then R1 is cooperative with R3 in D.

To illustrate the usage of these heuristics, let us consider some typical requirements for software systems. Suppose we plan to develop a billing information system, BILLSYS, with the following customer requirements:

• R_1: high reliability;
• R_2: low development cost;
• R_3: short time to deliver software;
• R_4: short time to diagnose and repair software;
• R_5: short system response time, and
• R_6: low maintenance costs.

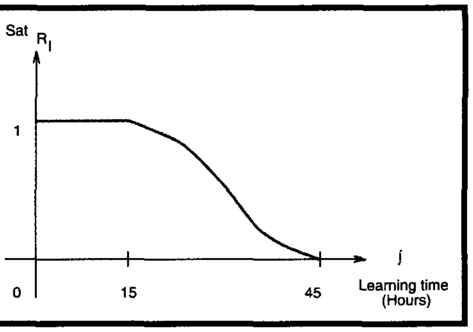


Fig. 2 The satisfaction function of requirement R_l

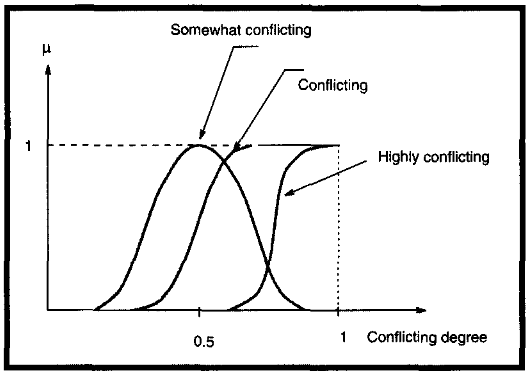


Fig. 3 An illustration of somewhat conflicting and highly conflicting requirements

High, *low* and *short* are fuzzy terms that can be characterized by fuzzy sets. There are conflicts among these requirements. Therefore, it is impossible to satisfy all requirements fully. As a result, trade-offs among them are desirable.

The initial qualitative specification of trade-offs among requirements are given by an experienced requirement analyst and customer (Fig. 4). Here "-" denotes a conflicting relationship between requirements and "+" denotes a cooperative relationship between requirements.

More relationships can be identified between requirements using the heuristic rules for reasoning. For example, using Heuristic rule 2, it has been inferred that R_1 and R_2 conflict with each other. This is because R_1 conflicts with R_3 and R_2 is cooperative with R_3. The refined description of the relationships between requirements is shown in Fig 5. The inferred relationships are represented by dashed lines and the original relationships are represented by solid lines.

Aggregating multiple fuzzy requirements

The overall system requirement for a system usually has multiple requirements. Fusing them into one overall requirement involves partitioning the requirements into groups and choosing suitable fuzzy aggregation operators. These operators combine requirements in each group and combine different groups. A set of cooperative requirements can usually be satisfied at the same time. Thus, a fuzzy conjunctive operator may be preferred for combining them.

Examples of fuzzy conjunctive operators include *MIN* and the algebraic product. Mutually exclusive requirements cannot be satisfied all at the same time. Only one can be satisfied. Therefore, fuzzy disjunctive operators can be used to combine them.

Examples of fuzzy disjunctive operators include *MAX* and the algebraic sum. Fuzzy compromise operators are usually used to aggregate conflicting requirements to achieve trade-offs. They do this by allowing compensation between requirements. The arithmetic mean is an example of a compromise operator.

Figure 5 shows that the requirements in the billing information system, BILL SYS, can be grouped as $G_1 : (R_1, R_4, R_6)$, $G_2 : (R_2, R_3)$ and $G_3 : (R_5)$. Requirements in each group are cooperative. Thus, they can be aggregated with a fuzzy conjunctive operator. We can use a compromise operator to combine all groups since trade-offs among them are desirable. Therefore, how well the overall customer requirement is satisfied by the billing system can be computed. This is done using the satisfaction degrees of individual requirements and fuzzy conjunctive and compromise operators.

Summary

Software requirement analysis and specification is an important activity in software development and maintenance. Without explicitly specifying customer's requirements, engineers do not know what to build and how to validate the software. This approach captures the elasticity of requirements using fuzzy sets. It helps to perform a trade-off analysis between requirements. It also facilitates formulating the overall requirement using the results of the trade-off analysis among conflicting requirements. Consequently, it can help create a better system objective that is satisfactory to customers and feasible to developers.

Reqts \ Reqts	R_1 High reliability	R_2 Low development cost	R_3 Short time to deliver software	R_4 Short time to diagnose and repair software	R_5 Short system response time	R_6 Low maintenance cost
R_1 High reliability						
R_2 Low development cost						
R_3 Short time to deliver software	−	+				
R_4 Short time to diagnose and repair software	+					
R_5 Short system response time		−	−			
R_6 Low maintenance cost	+				−	

Fig. 4 An initial description of relationships between requirements

Reqts \ Reqts	R_1 High reliability	R_2 Low development cost	R_3 Short time to deliver software	R_4 Short time to diagnose and repair software	R_5 Short system response time	R_6 Low maintenance cost
R_1 High reliability						
R_2 Low development cost	−					
R_3 Short time to deliver software	−	+				
R_4 Short time to diagnose and repair software	+	−	−			
R_5 Short system response time		−	−			
R_6 Low maintenance cost	+	−	−	+	−	

Fig. 5 Inferring relationships between requirements

Read more about it

• Davis, Alan, *Software Requirements: Analysis and Specification*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

• Hsia, Pei; Davis, Alan and Kung, David "Status report: Requirements engineering," *IEEE Software*, Vol. 10, no. 11, pp. 75-79, November 1993.

• Liu, Xiaoqing Frank and Yen, John, "An Analytic Framework for Specifying and Analyzing Imprecise Requirements," *Proc. of the 18th IEEE International Conference on Software Engineering*, Berlin, Germany, March 1996.

• Yen, John and Liu, Xiaoqing Frank, "Approximate reasoning about priorities of imprecise conflicting requirements," *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, Vol. 3, no. 2, pp. 53-73, June 95.

• Yen, John; Liu, Xiaoqing Frank and The, Swee Hor, "A fuzzy logic-based methodology for the acquisition and analysis of imprecise requirements," *The International Journal of Concurrent Engineering: Research and Application (CERA)*, Vol. 2, no.4, pp. 265-277, Dec. 94.

• Zadeh, L.A., "Fuzzy sets," *Information and Control*, Vol. 8, no. 3, pp. 338-353, 1965.

• Zimmermann, H.-J., *Fuzzy Set Theory and Its Applications*, Kluwer Academic, Boston, MA, 1991.

About the author

Xiaoqing Frank Liu is currently an assistant professor at the Computer Science Department in the University of Missouri-Rolla. He graduated with a PhD in computer science from the Texas A&M University at College Station in August, 1995. He has conducted research in software engineering, database systems, fuzzy logic and knowledge based systems since 1985. He has published about twenty referred journal and conference papers.