Abstract—Component-based approaches have acquired a prominent role in development of complex software systems. Successful reuse of existing components requires being able to first identify, and then distinguish among, functionally (near-) equivalent elements of large component collections. Similar components can be ranked using quality criteria; thus, some goal-oriented techniques attempt to quantify components quality by indicating valid ranges for their properties and behavior, like stability, latency and so on. Unfortunately, most current techniques yield non-robust ranges, and most tools do not allow architects to observe the range selection during the process. This paper presents a technique for sensitivity analysis of components discovery, built over fuzzy sets. A prototypical tool has been built, and use of the technique and tool are illustrated with an example. This iterative approach allows evaluators to compare “what if” scenarios for alternative component quality criteria, supporting requirements evolution without continuous expert support to recalibrate valid property ranges.

keywords - component discovery, fuzzy sets, sensitivity analysis

I. INTRODUCTION

A critical activity to successfully build software from a set of reusable and tested components is to discover potential candidates within a large collection; a poor selection will yield useless material to later component selection and composition stages [1]. The explosive growth of the component market makes almost impossible for architects to manually evaluate alternatives using only their experience and knowledge. The similarity of offered components means that mere comparison of functional attributes is not always enough to discriminate alternatives; components may differ widely in terms of reliability, latency, response time, and business aspects like cost and licensing. Therefore, the discovery process should be driven by not only functional aspects, but also using quality criteria and non-technical aspects. But quality is a fuzzy term, typically expressed in words, and, like Zadeh put so well [2], "humans have this remarkable capability to make rational decisions in an environment of imprecision, uncertainty, incompleteness of information and partiality of truth", but machines have neither.

Several strategies have been proposed to deal with imprecision in non-functional requirements (NFRs), by quantifying them with common metrics [3] to specify the desired value ranges for each NFR. We build on our previous work [4], which applied fuzzy sets to allow services membership in several categories at the same time but with different membership degrees.

Component discovery is not a static process, neither only executed at the beginning of the component-based development process, but instead it spans from the requirements stage to the maintenance phase [5]. As the market changes, our understanding of the project also changes, and therefore required quality aspects could be restricted or relaxed compared to previous stages. But what would be the real differences between of to keep the current decisions or to relax (restrict) them. What if, if we only relax just a little bit. We call to this task sensitive analysis. And we believe that this task needs to be supported by proper tools that help to understand what are the implications of to relax or restrict requirements of quality aspects. In this work we present a tool which allows evaluators to reason their relaxing or restricting decisions and to visualize the trade-off between different aspects, to compare recommendations before and after the change, and to rank again the complete component dataset according to these changes.

The reminder of this article is structured as follows: Section 2 provides the necessary background to understand the proposal, focusing on fuzzy theory; Section 3 presents the proposal; Section 4 illustrates the proposal with an example and give more details of the prototypical tool; Section 5 surveys some related work; Section 6 presents conclusions and future works.

II. BACKGROUND

Two main concepts need to be explained to provide a basic background to easier the explanation of the proposed approach.
A. Fuzzy sets and linguistic variables

In classical set theory, an object either is or is not member of a set, but membership in a fuzzy set is a graded, where e.g. an object $x$ may belong to a fuzzy set $A$ in a certain degree between 0 and 1 (values closer to 1 indicate higher degrees of membership).

Fuzzy sets were proposed by Zadeh [6]. Formally, a fuzzy set $A$ is defined by a set of ordered pairs, $A = \{(x, \mu_A(x)), x \in U, \mu_A(x) \in [0, 1]\}$, where $\mu_A(x)$ is a function (piecewise continuous or discrete) called membership function, that specifies the grade or degree to which any element $x$ in $U$ belonging to the fuzzy set $A$.

A fuzzy value is a fuzzy set which membership function is continuous and defined over the real numbers. There are several forms to represent fuzzy values; the triangular and trapezoidal shapes are most popular (Figure 1 shows a trapezoidal example). Depending of the type of the fuzzy number, we can need different parameters in order to define it: a triangular fuzzy value can be defined by a triplet $(a, b, c)$, but a trapezoidal fuzzy value can be defined by four parameters $(a, b, c, d)$. Typically these parameter values are decided upon by a group of experts or by consensus of the users.

On the other hand, linguistic variables can also be defined by fuzzy sets. A linguistic variable refers to the possible states are fuzzy sets assigned to relevant linguistic terms (e.g., “important”, “acceptable”, “not fast”, etc.).

For instance, if we need to match the goal “response time” with different component features, we can have three fuzzy sets: “fast”, “average”, and “slow”. According to this response value, we can return all the Web services that belong to the class that the consumer requires over a threshold (which could be also specified by the evaluator). Components that belong in some degree to a class may also belong to other classes, perhaps with different membership degrees. In Figure 1, the black arrow shows that a particular value is not binary classified as member of one class or another; instead, the “response time” of this component can be classified as “fast” in certain degree as well as “average” but in a different degree.

Fuzzy logic is a problem-solving approach that simulates the process of normal human reasoning under uncertainty. Fuzzy decision-making uses fuzzy logic to reach a crisp solution to a specific decision or control problem. Usually it involves three steps: fuzzification, fuzzy reasoning, and defuzzification.

B. Quality Attribute Utility Trees

One of the most used software architecture evaluation methods is the Architecture Tradeoff Analysis Method (ATAM) [7]. Its purpose is allowing to assess the consequences of architectural decisions in light of quality attribute and business requirements. The initial steps of the ATAM call for generating a “quality attribute utility tree”, which helps stakeholders to identify, prioritize and refine the most important quality attribute goals. The main advantage of utility trees is that they also support agreement on measurement scale during the attributes refinement process. For instance, in a particular scenario (finding a data storage component), evaluators might refine the “performance” attribute into the “latency” attribute; then decide to measure the values of the “latency” using milliseconds; and then, based on their own knowledge of the market and on their own experience, decide upon ideal and minimal/maximal acceptable values for each scenario. E.g. that “latency” should smaller than 200 [ms] (reserved value) and ideally only 100 [ms] (desired value).

III. Proposal

We propose a framework that semi-automatically supports the component discovery process at different stages of the project, validating whether some earlier recommendations become invalid (e.g. due to market changes), and allowing evaluators to explore nearby solutions by automatically relaxing or restricting initial evaluator inputs. Evaluators can then analyze positive and negative contributions on other requirements, and reasoning about the impact of the change (similarly to softgoal analysis [8]).

The framework is explained in the following subsections.

A. Specifying requirements and quality goals using the utility tree technique

The aim of this phase is gather requirements, specify their quality goals, and rank their relative importance.

Evaluators must identify the architectural concerns that the system must have. A concern is typically a non-functional aspect of the system. Most concerns can be mapped to non-functional requirements (e.g. “ease of use” to “usability”) and these can be mapped also to other measurable sub-requirement through specific metrics (e.g. “learnability time” ≤ 3 [hr], using as a testing mechanism “a set of exercises that a team must solve after a training”).

For each scenario several NFRs but only one functional requirement can be addressed. Evaluators generate an utility tree for each scenario, indicating the most important concerns that the system needs to address; for each concern they identify NFRs; and by refining each NFR, they obtain sub-NFRs that they must control.
As a brief example, consider the scenario “All operations are processed in fastest possible speed”. An important NFR is “performance”; refining this attribute could perhaps lead to decide that what actually matters is latency (see Figure 2); then, for each sub-attribute, its relative importance must be specified as “high”, “medium” or “low”.

Evaluators must also discern the relative importance of scenarios, using the same “high”, “medium” and “low” values. In the example, they could say that the scenario has “high” importance and that a component which offers access latency time to a data storage component between 100 ms and 200 ms is acceptable in “X” degree (see Figure 3). Values closer to 1.0 indicates that the solution is most acceptable.

B. Finding component sets that matches the initial requirement

The aim of this phase is to obtain, for each requirement or scenario, a component ranked list where each component belongs to some degree to an acceptable solution set because it satisfies the functional requirement as well as the NFRs in some degree.

1) Problem Formulation: For each functional requirement $FR = \{FR_1, ..., FR_N\}$, let $Q = \{q_1, ..., q_J\}$ be a set of $J$ sub NFRs (or quality goals). Let $D = \{D_1, ..., D_J\}$ be the desired and reserved (worst acceptable) value sets for each element of the set $Q$. Let $C = \{C_1, ..., C_M\}$ the set of $M$ functional-equivalent components satisfying the specific $FR$. Let $CC_m = \{c_{m1}, ..., c_{mJ}\}$ the current values that the component $m$ provides for each quality attribute. Even when cost is nor a quality attribute but is a non-technical aspect, in terms of simplicity, the reader must assume that the cost is the $J-th$ element of the quality attributes set.

Let $W = \{w_1, ..., w_J\}$ the set of importance weights that the evaluators assigned to each quality attribute for this particular problem. The weight is calculated assigning to each label a predefined value normalized by the sum of all the values assigned to the different aspects that the evaluators are considering. These relative importance labels were specified by the evaluators in the previous stage.

2) Evaluating potential solutions: Each component which satisfies the $FR_n$ and at least satisfies in some degree one quality attribute is considered as a possible solution because its membership degree to the acceptable solution set is greater than 0. To evaluate and compare these different solutions we use an utility function based originally on a classical work proposed on [9] (and recently also found on [10]). The utility of a potential solution $S_{mn}$ offered by the component $m$ for the particular functional requirement $FR_n$ is defined as

$$U(S_{mn}) = \sum_{j=1}^{J} w_j \delta(c_{mj})$$

(1)

where $w_j$ is the relative importance of the quality attribute $j$, $c_{mj}$ is the statistical value provided by the component $m$ for the quality attribute $j$ (obtained by a certifier entity) and $\delta(c_{mj})$ is the satisfaction level of the quality goal based on $c_{mj}$. A large value of $\delta$ indicates that the parameter $c_{mj}$ is closer to the desired value $D_j$ of the quality attribute $j$.

It is important to mention that the possible values that $\delta(c_{mj})$ may take depends on whether the reserved value is larger or smaller than the desired value required for the quality attribute $j$. For example, if the concern states that a 99.9% of availability is needed (desired value) and the reserved value is 99.0%, then the large the offered availability, the larger $\delta(c_{mj})$ will be. On the other hand, if the parameter is “time to learn to use the system”, with a desired value of 3 hours and a reserved one of 4 hours (reserved value larger than desired value), the smaller the offered time to learn, the larger $\delta(c_{mj})$ will be.

Formally, let the desired and reserved value of the quality attribute $j$ be $D_j$ and $R_j$ respectively; then we can compute $\delta(c_{mj})$ as follows:

If $R_j < D_j$, then

$$\delta(c_{mj}) = \begin{cases} 
0 & \text{if } c_{mj} < R_j \\
\frac{c_{mj} - R_j}{D_j - R_j} & \text{if } R_j \leq c_{mj} \leq D_j \\
1 & \text{if } c_{mj} > D_j 
\end{cases}$$

(2)

If $D_j < R_j$, then

$$\delta(c_{mj}) = \begin{cases} 
0 & \text{if } c_{mj} < R_j \\
\frac{R_j - c_{mj}}{R_j - D_j} & \text{if } D_j \leq c_{mj} \leq R_j \\
1 & \text{if } c_{mj} > R_j 
\end{cases}$$

(3)

Figure 4 shows the $\delta(c_{mj})$ for our data storage component example.
Because $\delta(c_{mn})$ is scaled to the unit interval ($0 \leq \delta(c_{mn}) \leq 1$, $j = 1..J$), there are no problems if the original range of the different concerns is totally different, because these concerns are mapped to a normalized space. On the other hand, for each functional requirement $FR_n$, the sum of the weights regarding its quality aspects is equals to 1, i.e. $\sum_{j=1}^{J} w_j = 1$, which allows to know which quality attributes are more important than others.

With these assumptions, the utility function of the offer $S_{mn}$ satisfies the unit interval restriction, $0 \leq U(S_{mn}) \leq 1$, where this function indicates in which degree this particular potential solution satisfies the required quality attributes required for $FR_n$ (in our case, scenario).

The output of this phase is a list of components ranked according to their utility function. This list is the potential solution set for the particular functional requirement $FR_n$.

3) Addressing multiple requirements at once: So far, we have been working only with one functional requirement at once; this would make the approach almost unusable because all the possible incompatibilities and trade-offs should be manually managed by the architect. Thus, at this point and without loss of generality, we introduce the utility function for the complete functional requirement set, defined as follows:

$$U(CS) = \sum_{n=1}^{N} v_n U(S_n)$$  \hspace{1cm} (4)

In this case, $v = [v_1, ..., v_N]$ represents the relative importance that evaluators assign to each functional requirement. $S_n$ is one of the higher ranked solution for the functional requirement $FR_n$, and $CS$ is the combinatorial solution that addresses the $N$ requirements (or at least a subset of them).

Also, for a set of functional requirements and their respective quality aspects, there is a set of hard constraints which cannot be broken; e.g. it would be quite strange that two totally different server applications be used to host a same application. If these rules are broken, then the complete solution is marked as an infractor, and architects can decide whether to eliminate it or not. Currently, the only restriction we support is the maximal number of components in the composition.

C. Reasoning the solution set by relaxing or restricting requirements

The potential solution set was computed using one rigid satisfaction function. The slope of this function ranges lineally between the desired and reserved value (see Figure 3). We propose here to alter this shape by relaxing or restricting the function.

Let the current rigid satisfaction function the “normal class”, $\delta(c_{mn})$. We relax this class into the “very flexible” (VF) and “flexible” (F) classes by changing the shape of the curve of the normal class (see equation (5) and (6) respectively assuming $R_j < D_j$), as well as we restrict the normal class into the “rigid” (R) and “very rigid” (VR) classes (see equation (7) and (8) respectively).

$$\delta(c_{mj})_{VF} = \begin{cases} 0 & \text{if } c_{mj} < R_j \\ \delta(c_{mj})^{1/4} & \text{if } R_j \leq c_{mj} \leq D_j \\ 1 & \text{if } c_{mj} > D_j \end{cases}$$  \hspace{1cm} (5)

$$\delta(c_{mj})_{F} = \begin{cases} 0 & \text{if } c_{mj} < R_j \\ \delta(c_{mj})^{1/2} & \text{if } R_j \leq c_{mj} \leq D_j \\ 1 & \text{if } c_{mj} > D_j \end{cases}$$  \hspace{1cm} (6)

$$\delta(c_{mj})_{R} = \begin{cases} 0 & \text{if } c_{mj} < R_j \\ \delta(c_{mj})^2 & \text{if } R_j \leq c_{mj} \leq D_j \\ 1 & \text{if } c_{mj} > D_j \end{cases}$$  \hspace{1cm} (7)

$$\delta(c_{mj})_{VR} = \begin{cases} 0 & \text{if } c_{mj} < R_j \\ \delta(c_{mj})^4 & \text{if } R_j \leq c_{mj} \leq D_j \\ 1 & \text{if } c_{mj} > D_j \end{cases}$$  \hspace{1cm} (8)

Figure 5 shows the new function shapes.

Besides of obtaining a component result set from the literal requirements, the evaluators can explore different alternatives by relaxing or restricting previous decisions, enriching the solution set by modifying the ranking punctuation of each presented solution and unhiding possible solutions that have become feasible. Thus, evaluators can decide “what if” they relax or narrow initial restrictions without additional effort.
In later stages, when the market changes, if a chosen solution breaks some restrictions, or new better solutions have appeared in the market, this solution will be marked as infractor, warning evaluators that the current selection is no longer valid. Due to better knowledge and project understanding at later phases, evaluators may wish to change relative importances, quality aspects, add or remove new requirements, etc. A key advantage of this approach is that they can simulate these changes and accept the new state, or rollback to the previous one.

IV. IMPLEMENTATION, EXPERIMENTS AND RESULTS

A. Implementation and dataset

A prototype system was developed to test the approach; it allows evaluators to: (1) specify their functional requirements, and for each one, to specify a set of non-functional requirements; (2) specify between functional requirements relative importance; (3) specify for each functional requirement the relative importance between their non-functional requirements; (4) find a set of ranked composed solutions which satisfy the initial set of requirements; and (5) restrict or relax the current solutions.

The prototype was implemented with Netlogo 4.1 and can run on any desktop computer; an extended version of the tool will be made available via Web.

The test data described a large collection of Web services, which are a special kind of components. We used a recently published QWS Dataset published in [11], which includes 2507 actual Web service implementations; for each one, quality characteristics were measured with a benchmarking tool. The dataset does not include pricing information, so we assumed (for now) that price behaves like any other non-functional requirement. Since in our previous work [4] we clustered data functionally using GHSOM, functional requirements were specified as keywords; this allowed to get a set of \( M \) functionally-equivalent components (services) that could be potential candidate if it satisfies at least one NFR restriction.

B. Illustrative Experiment

The objective of this experiment was to explore the growth of the discovery space size when customers relax requirements. In this example, a customer is searching for a Web service to get a weather prediction for a specific location for the next three days. We assume evaluators have built a quality tree selecting as quality aspects both reliability and response time; both are equally important. From the quality tree, evaluators extract the desired and reserved values for each quality aspect as well as their relative importance, and specify their requirements into the tool. Then, the tool calculates the normal function that represents for each non-functional requirement the membership function to the acceptable class. The threshold is set up as its default value (0.5).

Figure 6 shows the results obtained for this particular query. As we can appreciate, of the 11 potential and functional-equivalent weather Web services, only 7 are actually QoS-equivalent and need to be reviewed. Notice that some components are yellow painted, which means that although they actually belong to acceptable services, their membership degrees are too low (below 0.5) and were not considered as part of the final solution.

Table I shows some of the candidates. The first column indicates the Web service provider; the second column indicates the membership degree (e.g. *cweather* has a 0.95 of membership degree to the class "acceptable" (or normal) defined by the rule that the customer specifies; the third and fourth columns indicate the value for the quality concern and (in parenthesis) the membership degree to the specified range for each sub-NFR. Notice that some of the potential alternatives could have the same name (but totally different endpoint, and therefore probably are different Web services).

Do notice that, if evaluators relax the restriction of these two sub-NFRs to “very flexible” instead of the normal class, the set of potential solutions increases from 7 to 10 because the membership degree to the "very flexible" class overcome the threshold of 0.5 that we mentioned before.

![Figure 6. Result of experiment](image-url)

<table>
<thead>
<tr>
<th>provider</th>
<th>md</th>
<th>response time</th>
<th>reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOTSFastWeather</td>
<td>0.67</td>
<td>103 (0.75)</td>
<td>73 (0.94)</td>
</tr>
<tr>
<td>ndfdXML</td>
<td>0.54</td>
<td>396 (0)</td>
<td>67 (0.86)</td>
</tr>
<tr>
<td>cweather</td>
<td>0.95</td>
<td>49.43 (1)</td>
<td>73 (0.93)</td>
</tr>
</tbody>
</table>

^1^ [http://ccl.northwestern.edu/netlogo](http://ccl.northwestern.edu/netlogo)

^2^ [http://www.uoguelph.ca/~qmahmoud/qws](http://www.uoguelph.ca/~qmahmoud/qws)

^3^ [http://www.ifs.tuwien.ac.at/andi/somlib/](http://www.ifs.tuwien.ac.at/andi/somlib/)
V. RELATED WORK

Several approaches have been proposed to support the matching process between component features and customers' requirements and constraints. During the development process, requirements and restriction are refined, and new ones could appear, others disappear, and others could be relaxed or become more rigid.

Alves et al. [10] established the need to relax the requirements so customer can continually and easily negotiate and change them to reflect updated market offerings. They proposed a framework for component discovery that matched the set of operationalization goals that customer needs and managed the trade-offs among them. They also proposed a measurement strategy to quantify the satisfaction degree of the goals: for each operational goal, the evaluator define the acceptable interval (crispy values between the desired and worst acceptable values), and the lineal satisfaction function ranges over this interval. Then, the overall component satisfaction is obtained by aggregating all the importance weight and quality satisfaction pairs. Goals are refined with soft goals, and use an exploratory scenarios strategy to manage mismatches.

The main problem that we detect is the lack of flexibility of the satisfaction function, even when a different curve could be used. We agree that evaluators define the range of the acceptable interval (though problems could arise when evaluators do not use market information [4]), but we disagree to use a unique satisfaction function because this don't let evaluators to correctly relax or restrict some operational goals. One way to relax them is changing the acceptable interval, but this actually introduces other values to the range, giving them also a satisfaction value. A second way would be that each time the relaxing degree is changed for a particular aspect, the shape of the satisfaction function should be changed as well, to give more satisfaction degree to some part of the acceptable range (e.g. values closer to the desired ones). A third way would be modifying the importance weights associated to each quality attribute, but this would also change its critical importance.

In earlier work [12], inspired in the Gilb method used on software engineering as a strategy to quantify quality aspects and verify they have been satisfied by the implementation, we also defined a linear satisfaction function between the desired and the worst acceptable value. Of course, this function also suffers of lack of flexibility.

Cortellessa et al. [13] proposed to combine the approach Alves et al.'s approach [10] and their own earlier work [5], the Decision Support for Component-based Software framework (DEER). They automatized the component selection process in all the phases of a component-based development process. Their tool can also assist evaluators when requirements change allowing them to modify the range of the function of each NFR aspect. But evaluators cannot perform a sensitive analysis because they only use one class (one function). Instead, our approach automatically creates additional alternative functions around the function defined by the evaluators, which support to them, to find hidden neighborhood solutions.

Several manual approaches to component selection are extensively listed in [14].

Serban et al. [15], a component selection algorithm based on quality metrics and fuzzy clustering has been proposed that automated completely the selection process. Based on the metrics values of the component dataset, the algorithm starts with a minimal subset on independent requirements that need to be satisfied with a subset of components. For this particular subset of requirements, the components are arranged according to their similarity in two fuzzy clusters. The most representative candidate of each cluster is computed. A second algorithm decides which one of the two components is the best candidate, mostly based on the quality criteria values. The algorithm is iterative and stops until all the requirements are satisfied if that is possible. At the end, a set of components iteratively added are the proposal for the required requirements. A limitation of this work is that authors are working only with four non-technical criteria and for those criteria they are not using a quality standard. The aim of fuzzy at this works was clusters the components according to their functionality and they only use arbitrarily two clusters (for each time that these were recalculated). The disadvantage of this approach is works as a black box for the component evaluators (customer, software architects, designers, to name a few) and they do not support the tradeoff process that evaluators could make if they would have more information. On the other hand, this is not a quality driven approach, quality is slightly addressed, as a final decision criteria.

Simão e Blechior [16] proposed to use as consensus tool a quality guide that allows to gather from experts the satisfaction level of each refined goal. These data is treated as a Fuzzy Model for Software Quality Evaluation, and used to determine the relative importance (weight) of each attribute using fuzzy logic; the linguistic classes are computed from the data received of experts.

Cooper et al. [17] addressed the meaningful representation of uncertainty of non-functional attributes, and proposed using fuzzy logic to formally specify the non-functional part of components, clustering them from data obtained from a study performed with real components. This proposal did not address the ability to select components based on multiple attributes. Later works of the same authors does address selection based on multiple attributes modeling as an MCDM problem and solving using AHP, but focused on computing the possible set of candidates rather than being a reasoning model to allow architects or evaluators to evaluate their decisions’ trade-offs.
VI. FURTHER WORK AND CONCLUSIONS

The validity of this proposal has been tested with a real dataset of Web services, since Web APIs and Web services are special case of software components. Currently, we are developing a tool (Knowledge) to crawl, parse, cluster and act as a certifier entity regarding quality measurements.

Current work aims to build a sensitivity analysis tool enabling development team members to perform sensitivity analysis of design decisions and to visualize the impact of relaxing or restricting constraints.

Our goal is to allow architects to analyze “what if” scenarios at every stage; since architects win knowledge as the project progresses, they should be able to relax or restrict previous decisions, which may “unhide” components that have become more suitable than initial recommendations.

ACKNOWLEDGMENTS

This work was partially funded by FONDEF (grant D08i1155 ContentCompass), UTFSM DGIP (grant 24.08.58), and CCTVal (proy FB/22HA/10, proyecto Basal FB0821). Torres’ work was partially funded by UTFSM PIIC.

REFERENCES