Requirements Engineering (2023) 28:3–22
https://doi.org/10.1007/s00766-021-00368-y

ORIGINAL ARTICLE

# A negotiation support system for defining utility functions for multi-stakeholder self-adaptive systems

**Rebekka Wohlrab**[1] · **David Garlan**[1]

Received: 14 June 2021 / Accepted: 24 November 2021 / Published online: 11 January 2022


**Abstract**
For realistic self-adaptive systems, multiple quality attributes need to be considered and traded off against each other. These quality attributes are commonly encoded in a *utility function*, for instance, a weighted sum of relevant objectives. Utility functions are typically subject to a set of constraints, i.e., hard requirements that should not be violated by the system. The research agenda for requirements engineering for self-adaptive systems has raised the need for decision-making techniques that consider the trade-offs and priorities of multiple objectives. Human stakeholders need to be engaged in the decision-making process so that constraints and the relative importance of each objective can be correctly elicited. This paper presents a method that supports multiple stakeholders in eliciting constraints, prioritizing quality attributes, negotiating priorities, and giving input to define utility functions for self-adaptive systems. We developed tool support in the form of a blackboard system that aggregates information by different stakeholders, detects conflicts, proposes mechanisms to reach an agreement, and generates a utility function. We performed a think-aloud study with 14 participants to investigate negotiation processes and assess the approach's understandability and user satisfaction. Our study sheds light on how humans reason about and how they negotiate around quality attributes. The mechanisms for conflict detection and resolution were perceived as very useful. Overall, our approach was found to make the process of utility function definition more understandable and transparent.

**Keywords** Self-adaptive systems · Quality attributes · Utility functions · Analytic hierarchy process · Blackboard architecture · Requirements prioritization · Requirements negotiation · Non-functional requirements

## 1 Introduction

For self-adaptive systems, multiple quality attributes (such as performance, availability, and security) need to be considered and traded off against each other. These quality attributes are often encoded in a *utility function*, i.e., a single aggregate function whose expected value should be maximized by the system [22, 31, 35, 56]. In self-adaptive systems, utility functions are typically used by automated planning mechanisms to identify the relative costs and benefits of alternative strategies. In related work, utility functions are often defined as the weighted sum of relevant objectives [12, 21, 26, 61]. For most approaches using utility functions, it is simply stated that they should be manually defined, but little guidance for this task is provided [35, 61]. While the issue of utility function definition is prevalent in a variety of domains (e.g., in environmental sciences and consumer research [11, 30]), we focus on the domain of self-adaptive systems in this paper. In this domain, it is particularly challenging to correctly identify utility function weights and consider trade-offs between multiple quality attributes, as reported in the research agenda for requirements engineering for self-adaptive systems [56].

Moreover, self-adaptive systems often have multiple stakeholders (e.g., end users or business owners) whose preferences need to be consolidated to identify the overall relative importance of each objective [55]. Decision-making techniques are needed to help stakeholders prioritize and negotiate quality attributes and determine appropriate utility function weights [56]. Besides determining utility function weights, it is also important to capture constraints on specific quality attributes [5, 9, 70]. For instance, in practice, a self-adaptive system would be unsatisfactory if it ran out of energy while completing a task. For this reason, utility functions are typically subject to a set of hard constraints [70] that indicate the acceptable

✉ Rebekka Wohlrab
wohlrab@cmu.edu

[1] Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA, USA

values for quality attributes (e.g., that the battery charge should always be above a certain value). In practice, it is difficult for stakeholders to collect and consolidate preferences and constraints, reach an agreement, and define a utility function that can be used by a self-adaptive system [56].

In this paper, we present a lightweight tool-supported method for utility function definition for multi-stakeholder self-adaptive systems. The method is based on the Analytic Hierarchy Process (AHP) [54] and the Delphi technique [34]. It supports stakeholders in prioritizing quality attributes, specifying constraints, negotiating priorities to reach an agreement, recording rationales and comments, and giving input to define utility functions. For utility functions, we use the weighted sum approach, as it is lightweight and commonly used in related work [12, 16, 26, 61]. It assumes that the weighted quantity of one quality attribute can be traded off (or "substituted" [1]) with another one. Moreover, our approach supports the elicitation and consolidation of constraints, conflict detection, and mechanisms to help stakeholders reach a consensus when conflicting constraints occur. We created tool support in the form of a blackboard system to help stakeholders collect relevant information and process it to arrive at a final utility function and constraints.

The paper is based on a previous conference publication [71] and has been extended by a more detailed description of the method, as well as developed tool support, including mechanisms for constraint specification, conflict detection, and suggestions for (semi-)automatic resolution mechanisms. Moreover, we present findings from a think-aloud study with 14 participants evaluating the approach's understandability and user satisfaction. Our findings indicate that the explanations provided by the tool, as well as the conflict resolution mechanisms, help to define a utility function in a transparent and understandable way with traceability to the initial user input. Our data suggests that our participants were generally satisfied with the tool support, although refinements to the usability are needed to increase the tool's maturity further.

The remainder of this paper is structured as follows: Sect. 2 describes our research method. Section 3 presents our AHP-based approach for utility function definition. In Sect. 4, we describe the tool support we implemented. The example system used in our think-aloud study is presented in Sect. 5. Section 6 presents the findings of our study. In Sect. 7, we describe related work. We discuss our findings in Sect. 8 and conclude the paper in Sect. 9.

## 2 Research method

We developed our contributions in several iterations based on an informal literature review, an investigation of related methods, our previously proposed approach for utility function definition [71], and internal discussions. We motivate our design decisions and the underlying reasoning in the sections where we present our approach and tooling.

To evaluate our approach, we performed a think-aloud study [43]. The think-aloud method [19] is a widely used technique to investigate problem-solving processes and participants' cognitive models [42, 65]. The main idea is that participants make spoken comments about their thoughts while working on a task. Think-aloud protocol analysis has previously been used by software engineering researchers, for instance, when evaluating user interfaces and understanding how developers perform and reason about tasks [42, 53, 69]. In the context of this paper, we decided to perform a think-aloud study because we were interested in the cognitive processes of utility function definition, negotiation, and conflict resolution.

For our think-aloud sessions, we prepared a series of tasks to be performed by the participants. The participants were asked to think-aloud while performing their work. We also conducted a short post-task interview in which we asked questions about the participants' experiences with the tool and encouraged them to think-aloud while answering.

We conducted the study with 14 academic participants who had an understanding of complex, software-intensive systems to analyze how they reason about utility functions, constraints, and preferences. We aimed to understand the applicability and understandability of our approach, as well as our participants' satisfaction levels with the employed mechanisms. Our think-aloud study and its findings will form a basis for further empirical studies focusing on the approach's applicability to real-world systems.

We were especially interested in how our participants experienced the approach, how they negotiated, and whether any tool mechanisms were challenging to understand. Assessing the mechanisms' understandability allowed us to analyze the mental models that human users develop when working with the negotiation support system, which can lead to better insights for the future development of approaches for utility function definition and stakeholder negotiation. Apart from understandability, we also focused on user satisfaction to investigate our participants' expectations and areas of improvement. Our research questions were as follows:

**RQ1:** How understandable are the blackboard system's resolution mechanisms?

**RQ2:** How satisfied are users with the blackboard system's output?

In the following, we elaborate on the study design, participant selection, data collection, data analysis, and threats to validity.

Study Design: The think-aloud sessions started with an introduction to the problem domain, where the facilitating researcher explained an example system that we used in the remainder of the study (more details will be provided in

Sect. 5). The goal was to ensure that the participants were aware of the system's context and the objectives of the study. The introduction was followed by a learning phase, in which the participants were given a role description of a stakeholder and were asked to try out the tool, enter constraints and preferences, and explore information on the blackboard. Conflict resolution was supported by the tool's mechanisms and negotiation was performed using the chat. We had previously determined role descriptions for all stakeholders, so that the interviewing researcher could adopt the role of another stakeholder and have a chat conversation with the participant in the negotiation phase.

Finally, we leveraged methods to elicit our participants' mental models of utility function definition. Following Hoffman et al.'s suggestions to evaluate solutions for explainable AI [32], we decided to conduct glitch detector tasks (in which people identify things that are wrong in a system/explanation) and prediction tasks (in which users are asked to predict a system's results and explain their predictions). These tasks help to understand whether the current system or explanation is understandable and in line with what participants would expect. The tasks were related to an estimation of the final weights of the utility function, participants' understanding of constraint resolution, concordance, and consistency. For instance, one of the tasks involved showing the participants a bar chart with non-concordant preferences for which a wrong utility function was generated. The participants should indicate whether or not they expected the preferences to be concordant and what weights they would expect the final utility function to have.

In the post-task interviews, we presented a list of Likert-scale questions to our participants and asked them to answer on a scale from "strongly disagree" to "strongly agree." Moreover, we asked about the experience of using the tool, difficulties, strategies for negotiation, and suggestions to develop the tool further. More details about the study procedure, as well as the material we used during the study, can be found online[1].

Participant Selection: We selected 14 participants with an understanding of complex, software-intensive systems (e.g., robotics or software systems). Several participants had worked with robot planning applications before and were aware of the challenge of defining utility functions. We selected participants with different backgrounds and roles, including four university faculty members, one researcher involved in industrial projects, two technical staff members/researchers, six PhD students, and one business office staff member.

Data Collection: All sessions were conducted via a video conferencing platform and took between 46 and 84 minutes, with an average of 64 minutes. The audio of the think-aloud sessions was recorded and transcribed for easier data analysis. Moreover, we collected tool data and observations during the sessions.

Data Analysis: We performed data analysis using QualCoder 2.4 [15]. QualCoder is a tool for qualitative data analysis of text, images, audio, and video. We performed *coding*, following Creswell's guidelines for qualitative analysis [14], and applied an editing approach. Our initial set of codes was based on the research questions and the predefined set of tasks. Two examples of these *a priori codes* are "understandability" and "concordance of preferences". The codes were refined, new codes were added, and several codes were merged during the analysis. To identify the research findings to report on in this paper, we went through the codes to analyze relationships and group them into categories. Our findings are reported in Sect. 6.

## 2.1 Threats to validity

We identified several threats to internal, construct, conclusion, and external validity. Internal validity/credibility: Threats to internal validity or credibility were partially mitigated by providing rich descriptions describing the contexts of statements in the think-aloud sessions. The decision to conduct a think-aloud study helped us to not limit ourselves to a fixed set of factors (as in a survey) but explore potentially confounding factors. Collecting data based on the transcripts of our sessions, tool data, and answers to glitch detector/prediction tasks helped us to triangulate different sources of information and elicit participants' mental models. What should be noted is that the participants of the study were given a description of their roles and asked to act according to that description. Studying utility function definition and stakeholder negotiation in a real-world context would likely lead to different findings. For instance, stakeholders might fight for their positions more than in our example scenario. In Sect. 8, we discuss further implications of this threat. We aimed for high transparency both when it comes to the explanation of our research method and the description of our findings. Using quotes ensured that findings can be traced back to statements from the think-aloud sessions which strengthens our findings' credibility.

Construct validity: Construct validity is concerned with how well our measures are suited to study the phenomenon under study. In our case, it was central to establish a common terminology with the participants, e.g., when it comes to terms like negotiation, consensus, priorities, or agreement. For this reason, we spent five to ten minutes at the beginning of the sessions to clarify the context of the study and ensure that all participants' questions were answered. We provide information about the introductory part of the study in the external document with our study's material[1].

[1] https://doi.org/10.6084/m9.figshare.17019125.v1

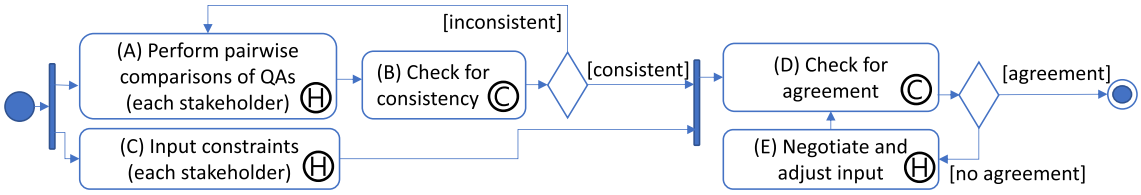


**Fig. 1** Our utility function definition method. Activities marked with (H) are performed by a human user, whereas activities marked with (C) are performed semi-automatically

Evaluation apprehension is another threat to construct validity and relates to participants trying to appear intelligent or good in the eyes of the researcher. It would be problematic to confound the effect of a treatment with apprehension. We acknowledge the issue of evaluation apprehension, although we investigate utility function definition in an exploratory fashion rather than intending to arrive at the finding that our tool (as a treatment) would lead to any measurable changes.

Conclusion validity: Conclusion validity focuses on the extent to which the findings in this paper are reasonable. It is especially concerned with whether we found relationships in our data that do not exist or whether we missed relationships that should have been found. While we did not aim to arrive at statistically significant conclusions in this study, conclusion validity is still relevant for our think-aloud study. The degree of reliability might have been compromised by the fact that we collected data from 14 participants in sessions of an average length of 64 minutes. Collecting data from an even larger number of participants would have led to a larger amount of information and the potential detection of further findings. To mitigate threats to reliability, we aim to be transparent about our research method and provide information about our study design material as external documents. We thoroughly discussed and refined the study material over several weeks to avoid issues related to a potentially incoherent structure or poor question-wording.

External validity: The study reported in this paper does not have broad generalizability as its goal, but rather presents an in-depth think-aloud study focusing on practical experiences with utility function definition. We selected the participants of this study based on their knowledge of complex, software-intensive systems, which is why the findings of this study are not necessarily transferable to other populations. However, involving participants with different roles helped us get a variety of perspectives on the topic and strengthen external validity.

One central threat is the presence of the main researcher who has both assumed a central role when developing the tool and facilitated the sessions. This threat related to reactivity might entail that our participants responded more positively because they knew that we were evaluating our tool. To mitigate this threat, we stressed that the participants should openly share their thoughts and that suggestions for improvement were especially welcome. Our results indicate that the participants followed these instructions and freely shared points of criticism, as 40% of the participants stated that the tool was not easy to use and suggested aspects to improve. A potential way to mitigate this threat further is to involve an independent group of researchers performing the same study. To facilitate the replication of our study, we provide supplementary material online[1].

## 3 A method for defining utility functions

Figure 1 shows the steps of our method for utility function definition, which was previously published in [71]. The method can be used either for the initial definition or the refinement of a utility function, in case stakeholders' preferences evolve. We assume that the involved stakeholders are aware of the quality attributes under consideration and know how they can be measured. For instance, stakeholders might be concerned with speed as a quality attribute (indicating how fast a system arrives at its target destination), safety concerns (penalizing collisions with objects), and energy consumption (indicating the battery charge). The leftmost steps in Fig. 1 are performed individually by each stakeholder. The guard conditions refer to whether an AHP matrix is consistent and whether an agreement has been reached. Each step is labeled with the paragraph of this section in which it is described.

Human stakeholders participate in all steps of the method shown in Fig. 1. The activities labeled with Ⓗ are manually performed. For the activities marked with Ⓒ, we developed tool support that performs checks, gives feedback to human stakeholders, and asks for input if needed. Our approach is based on the creation of a matrix that captures pairwise comparisons of quality attributes (A). When checking for consistency (B), the transitive property of pairwise comparisons is crucial [54]. In this context, consistency entails that if quality attribute X is preferred over quality attribute Y and Y is preferred over quality attribute Z, it must follow that X is also preferred over Z. The check for agreement in step (D) is concerned with the concordance of pairwise comparisons made by several stakeholders. For concordance, we compare the rankings of quality attributes (indicating which attribute

**Table 1** AHP judgment/preference options with numerical values [54]

| | |
|---|---|
| Extremely preferred | 9 |
| Very strongly preferred | 7 |
| Strongly preferred | 5 |
| Moderately preferred | 3 |
| Equally preferred | 1 |
| Intermediate values | 2, 4, 6, 8 |

**Table 2** Example of an AHP matrix

| | Safety | Speed | Energy consumption |
|---|---|---|---|
| Safety | 1 | 7 | 9 |
| Speed | $\frac{1}{7}$ | 1 | 1 |
| Energy consumption | $\frac{1}{9}$ | 1 | 1 |

is considered most, second, ..., and least important) and analyze how strongly different stakeholders' rankings agree with each other. Moreover, it is identified whether the specified constraints agree or conflict with each other. In the final step (E), our method supports stakeholders in negotiating and adjusting their input preferences and constraints.

*(A) Perform pairwise comparisons:* For the prioritization of quality attributes, we use the AHP, which is especially useful when subjective, abstract, or non-quantifiable criteria are relevant for a decision [54]. A central part of the AHP is to elicit stakeholders' priorities of different objectives in pairwise comparison matrices, which are positive and reciprocal (i.e., $a_{ij} = 1/a_{ji}$). For utility functions, we are interested in the degree of preference of one quality attribute over another, with the goal of increasing the overall utility of a system. Verbal expressions are used for these pairwise comparisons (e.g., "*I strongly prefer X over Y*"). Table 1 shows how the verbal expressions correspond to numerical values. When working with our tool, users are not required to create or understand AHP matrices. It is sufficient to perform pairwise comparisons and indicate their preferences.

For a robot planning problem, Table 2 shows an example of an AHP matrix with the attributes safety (expected number of collisions), speed (duration of a mission), and energy consumption (consumed watt-hours). In the example, safety is *very strongly* preferred over speed (7) and *extremely preferred* over energy consumption (9). Speed and energy consumption are *equally preferred*.

The relative priorities of the quality attributes can then be calculated using the principal eigenvector of the eigenvalue problem $Aw = \lambda_{\max} w$ [54]. $A$ is the matrix of judgments and $\lambda_{\max}$ is the principal eigenvalue. For the matrix in Table 2, the principal eigenvalue is $\lambda_{\max} \approx 3.01$. A corresponding normalized eigenvector to $\lambda_{\max}$ is $(0.8, 0.1, 0.1)^T$, which corresponds to the relative priorities of the quality attributes. A priority indicates the importance of a quality attribute with a value between 0 and 1, where a priority of 0 indicates that the quality attribute is not important at all. The relative priorities always sum up to 1 (given that they originate from the corresponding *normalized* eigenvector).

Utility functions are often used by automated planners to calculate the optimal plan for a self-adaptive system. The utility function for a plan $p$ can be defined as $U(p) = 0.8 \cdot \mathsf{utility}_{\text{safety}}(p) + 0.1 \cdot \mathsf{utility}_{\text{duration}}(p) + 0.1 \cdot \mathsf{utility}_{\text{energy}}(p)$ . $\mathsf{utility}_{\text{safety}}(p)$ is related to the expected number of collisions when executing the plan, $\mathsf{utility}_{\text{duration}}(p)$ captures the utility with respect to the duration of the plan, and $\mathsf{utility}_{\text{energy}}(p)$ is concerned with the consumed watt-hours. The preference of a quality attribute can often be described with a sigmoid function defining an interval for the quantity that is considered as good enough and an interval for the quantity that is insufficient [49]. Appropriate methods need to be selected to elicit these thresholds and define quality attributes' preference functions.

(B) Check for consistency: AHP matrices can be checked for consistency. A matrix is consistent if $a_{jk} = a_{ik}/a_{ij}$ for $i, j, k = 1, \ldots, n$ [54]. Saaty proved that a necessary and sufficient condition for consistency is that the principal eigenvalue of A be equal to $n$, the order of A [54]. He defined the consistency index CI as $(\lambda_{\max} - n)/(n-1)$. For our example in Sect. 3, CI is 0.004. To compare consistency values, Saaty also calculated the *random consistency index* RI by calculating CI for a large number of reciprocal matrices with random entries [54]. For a $3 \times 3$ matrix, the average random consistency index was 0.58. According to Saaty, the consistency ratio $CR = CI/RI$ shall be less or equal to 0.10 for the matrix to be considered consistent [54]. In our example, the consistency ratio is 0.01. If the condition for consistency is not fulfilled, stakeholders are required to refine their AHP matrices. The matrix can be automatically analyzed to point out the triples of quality attributes $\mathrm{QA}_i$, $\mathrm{QA}_j$, and $\mathrm{QA}_k$ where $a_{jk} \ll a_{ik}/a_{ij}$ or $a_{jk} \gg a_{ik}/a_{ij}$.

(C) Input constraints: Besides determining stakeholders' preferences, it is often important to elicit constraints when developing real-world systems [5]. These constraints cannot be traded off against other quality attributes, but need to be fulfilled in any case [71]. Our tool supports the specification of such constraints. We support both lower bound and upper bound constraints that can be associated with a real number (e.g., stating that the speed should be at least 2.0 m/s or at most 1.0 m/s). Moreover, stakeholders are asked to add a rationale explaining underlying reasons.

(D) Check for agreement: One of the aspects when checking for agreement is to identify conflicts in stakeholders' constraints. Conflicts occur when a lower bound constraint for a specific quality attribute specifies

an interval that does not overlap with an upper bound constraint's interval for the same quality attribute. For instance, stating that speed should be at least 2.0 m/s is in conflict with specifying that it should be at most 1.0 m/s. These conflicts need to be resolved in step (E). Besides conflict resolution, it is also possible that there exists an overlap between constraints, so that one constraint supersedes another. In these cases, both constraints are lower bound (or upper bound) constraints. For instance, if one stakeholder requires speed to be at least 2 m/s and another one requires speed to be at least 3 m/s, the former constraint would be superseded by the latter (since 'speed at least 3' is a stronger constraint).

For users' preferences, we use another check for agreement. We consider the rankings of $n$ quality attributes by $k$ stakeholders (where each quality attribute's rank is a number between 1 and $n$). The lower the rank, the more important is the quality attribute for a specific stakeholder. If all stakeholders rank energy as the most important quality attribute (rank 1), its ranking would be $1k = k$, the lowest possible ranking. For $QA_i$, the sum of ranks by all stakeholders is $R_i$, and the mean value of these ranks is $\bar{R} = \frac{1}{n}\sum_{i=1}^{n} R_i$. If the stakeholders' rankings do not agree, we can assume that the sums of ranks of several quality attributes are approximately equal [38]. It is therefore natural to consider the sum of squared deviations from the mean values of ranks $S = \sum_{i=1}^{n}(R_i - \bar{R})^2$ [38]. The maximum possible value of S is $k^2(n^3 - n)/12$ [38]. Kendall's concordance coefficient, describing the agreement of rankings in a [0,1] interval, is therefore: $W = \frac{12S}{k^2 \cdot (n^3 - n)}$ [38]. If the concordance coefficient is at least 0.3, the agreement is at least at a moderate level.

(E) Negotiate and adjust input: In case an agreement is not reached, a tool-supported negotiation and reprioritization phase starts. When it comes to conflict resolution for constraints, several options are suggested by the tool. The typical options for a user are to drop their constraint, decide based on stakeholders' authority levels (which allows the stakeholder with the highest authority to decide to drop another constraint), or negotiate in the chat.

For preferences, another negotiation and adjustment approach is used. To aggregate AHP matrices, the "most recommendable aggregation technique" is to calculate the weighted arithmetic mean of individual priorities (AIP) [46]. Stakeholders' priorities can be weighted differently, as their influence and stake may differ. In our approach, stakeholder authority levels can be defined to indicate their authority.

To resolve conflicts, we adapt the *Delphi technique* [34] for remote consensus building. The Delphi technique can be used "to seek out information which may generate a consensus", "to correlate informed judgments on a topic spanning a wide range of disciplines", and "to educate the respondent group as to the diverse and interrelated aspects of the topic" [34]. In our adapted version of the Delphi technique, interactive tooling is used to present the nature of the conflict(s), give participants a transparent overview of each other's preferences and constraints, and explain potential solution strategies (see Sect. 4.4 for details). Users can also declare that they do not want to indicate any preferences. Participants are encouraged to use the chat feature to discuss underlying objectives and arrive at an agreement. Comments and rationales are presented to the participants and the input can be revised, both with respect to constraints and with respect to preferences. The resulting utility function is a weighted sum of the objectives, where the final weights are the participants' aggregated weighted priorities (using AIP).

## 4 A negotiation support system for utility function definition

We developed our system for utility function definition based on the blackboard architecture pattern. The following sections introduce the system, starting with its architecture in Sect. 4.1.

The system follows the approach described in Sect. 3. The initial steps are concerned with the collection of user input (based on the pairwise comparison for AHP and the specification of constraints). Afterward, an automated 'consolidation agent' checks for agreement, identifies conflicting and superseded constraints, and feeds that information back to users. Moreover, the system supports negotiation and input adjustment. When an agreement has been reached, the resulting utility function is shown and explained to users.

Users interact with a web application and user interface that are described in Sect. 4.2. We present how our consolidation agent processes and consolidates stakeholders' input in Sect. 4.3, followed by a description of conflict resolution mechanisms in Sect. 4.4.

### 4.1 Blackboard system implementation

The blackboard architecture pattern is a software architecture pattern that was initially used for speech recognition [20] and has been applied in a variety of domains [13]. The pattern is based on the metaphor of several experts or agents looking at a blackboard, analyzing its current state, and adding information to it. These agents add and refine information on the blackboard until a problem has been solved. The pattern allows for diverse problem-solving techniques and flexible representation of information [13]. We found the pattern to be applicable for the problem of utility function

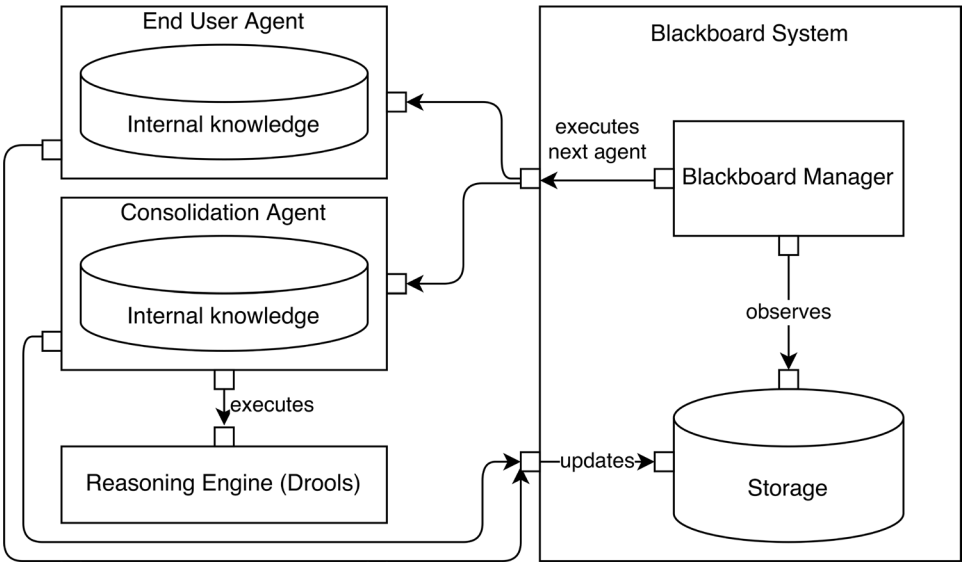


**Fig. 2** Overview of the architecture of the utility function definition system

definition where multidisciplinary stakeholders need to collect a variety of information and create a common utility function for a given system. The blackboard's agents can be human stakeholders or can be automated.

Figure 2 shows an overview of the architecture of our utility function definition and negotiation system. It consists of the blackboard system and several agents. To interact with human stakeholders, we developed an end user agent (implemented as a Vaadin 14 web application [27]). Moreover, we created a consolidation agent that processes and aggregates information. The consolidation agent evaluates the blackboard's status, detects and explains conflicts, and determines the utility function weights. It is described in further detail in Sect. 4.3.

The blackboard system consists of a blackboard manager component and a database to store information. The storage is implemented as a MySQL database. The information in the database is stored in "facts" and we distinguish between constraint facts, preference facts (for stakeholders' preferences), definition facts (establishing common definitions of the fact types), authority facts (indicating the stakeholder authority level for a quality attribute), as well as utility facts (indicating the final weights of the utility function). Facts can be linked to each other with superseded, removed, or parent relationships. These relationships between facts help to establish and maintain traceability.

The blackboard manager observes the storage and executes the next agent based on observed changes. It pushes a message to the next agent to indicate that it is its turn to update the blackboard system. The next agent is selected based on their authority levels and depending on what has changed in the system. For instance, if facts have been added and a human user has requested to generate the utility function, the consolidation agent is executed. If the consolidation agent needs information to resolve a conflict, it adds that information to the storage. The blackboard manager, in turn, sends a message to an end user agent about the nature of the conflict and requests information.

### 4.2 Web application and user interface

We used Vaadin 14 to implement the web application for the end user agent [27]. Vaadin supports several principles of user interface design out of the box, e.g., by providing control components, ensuring that designers choose appropriate color schemes, and supporting straightforward mechanisms to embed information or error dialogues [25, 27]. For our particular user interface design, we used the dashboard design pattern [68]. Our dashboard helps users to get an overview of the current state of the utility function definition process. The dashboard was enriched with forms (located in a bar on the left side of the screen) to allow users to provide input using control elements (i.e., sliders, text fields, drop-down lists, and chat/log textfields [25]). The advantage of applying the dashboard pattern is that users are not required to click many times to navigate through a (potentially complex) navigation structure. Overall, we relied on common elements for interface design [25, 68] that our participants were familiar with. What should be noted is that we developed the user interface on a device with a resolution of 1920x1080 pixels. The user interface is not as easy to use on smaller devices, which might motivate the need to redesign parts of the interface when using the system on other devices in the future.

Figure 3 depicts a screenshot of the user interface. The dashboard shows all information and blackboard facts at once. The user can input their preferences using the editor in the top-left corner ①, where sliders are provided for

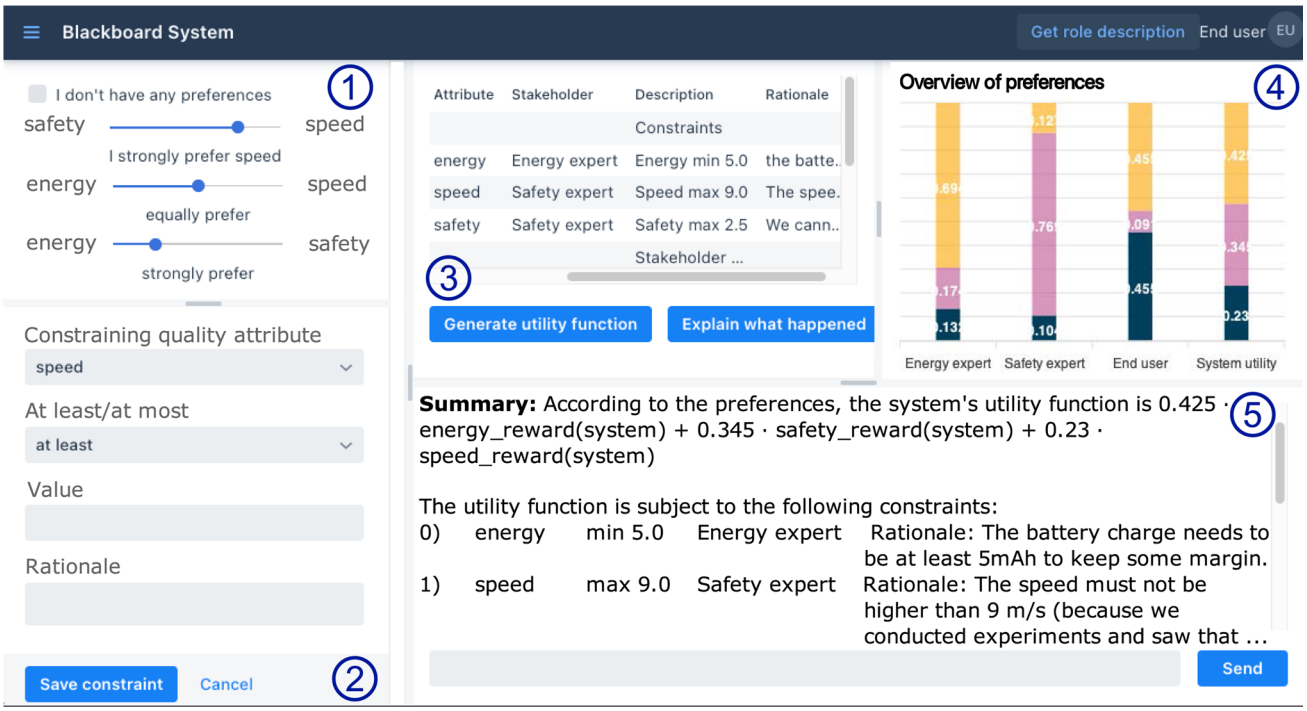


**Fig. 3** Screenshot of the utility function definition system

pairwise comparisons of the quality attributes. The sliders have a pin with an initial position at the center (indicating that both attributes are equally preferred) that can be moved toward the left or right to indicate the preference of one quality attribute over another. Rather than filling in an AHP matrix, sliders help users to focus on the pairs of quality attributes to be compared and visualize how strong the preference is by supporting different positions of the sliders' pins. One advantage of using sliders is that stakeholders do not need to work with AHP matrices or numerical values, which greatly reduces the mathematical complexity they need to deal with. Moreover, a constraint editor is provided, allowing the specification of constraints that state that the value of a quality attribute shall be at least or at most a certain value ②. It is also possible to define a rationale for a constraint. At the center, the blackboard facts are displayed. A list of facts shows all currently inserted facts, including the attribute, stakeholder, description, and rationale ③. Below the list, two buttons are provided allowing users to generate a utility function and to request an explanation of what happened. Preferences are visualized as bar charts to help end users get an overview of the priorities that the stakeholders assign to the quality attributes ④. Finally, the bottom right part is a chat/log window ⑤, indicating the current state of the consolidation, but also allowing users to send messages to each other (e.g., in the negotiation phase).

The web application is used to support the steps shown in Fig. 1, i.e., performing pairwise comparisons (using the sliders), inputting constraints (using the form), getting feedback from the consolidation agent regarding consistency and agreement issues, as well as negotiating and adjusting the initial input. Depending on the state of the blackboard system, the user is prompted to give input to resolve conflicts and reach an agreement. When an agreement has been reached, a summary is shown to users, describing what the final weights of the utility functions are and what constraints the utility function is subject to. Moreover, it is possible to view information related to the constraints by clicking on them in the list of facts. Figure 4 shows an example of information for an end user constraint, requiring speed to be at least 2.0. It is shown that a fact was removed due to a conflict with this constraint, namely the safety expert's constraint requiring speed to be at most 1.0. The number line below shows the speed values that are allowed according to the two constraints and it can be seen that there is no overlap between the two constraints' lines. For superseded constraints, a similar explanation is generated by the consolidation agent.

### 4.3 Consolidation agent

The consolidation agent's role is to detect and explain conflicts that arise in the constraints or preferences of multiple stakeholders, as well as to generate the utility function weights. The consolidation agent uses the reasoning engine Drools [50]. Drools has previously proven

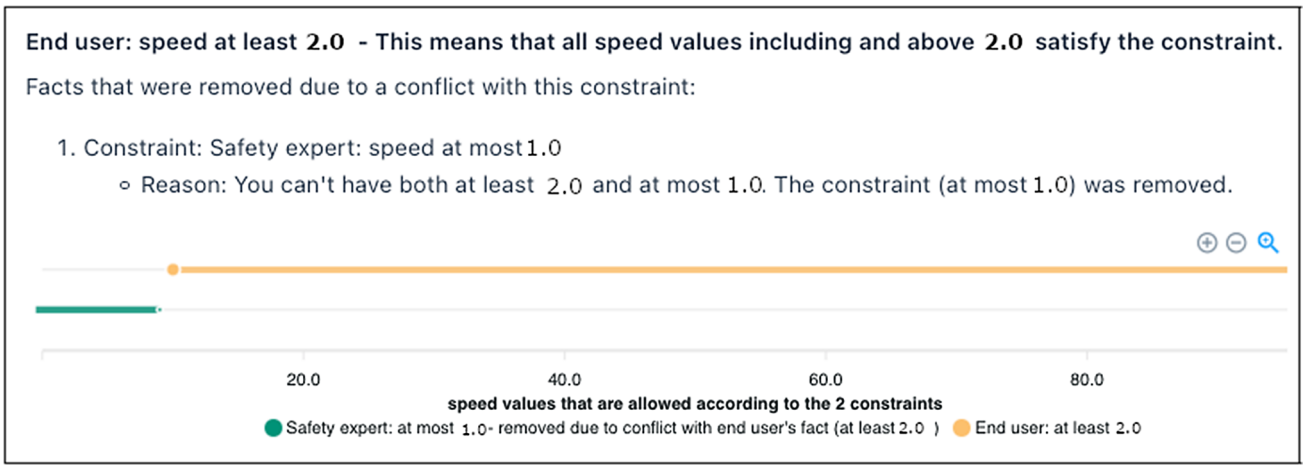


**Fig. 4** Information about an end user constraint (speed at least 2.0)

**Table 3** Overview of Drools rules for our blackboard system

| | |
|---|---|
| 1. Superseded constraints | If one constraint for a specific quality attribute implies another (not superseded or removed) constraint, the latter constraint is added to the superseded facts of the former. |
| 2. Conflicting constraints | If one constraint for a specific quality attribute is in conflict with another (not superseded or removed) constraint, the latter constraint is added to the removed facts of the former (provided that the latter constraint's stakeholder's authority level is lower than the other one's). |
| 3. Concordance check | If the concordance value of the current preference facts is less than a certain value (0.3), an explanation is generated to inform stakeholders about non-concordant preferences and present them with options. |
| 4. Create utility facts | If there exists a preference fact for a quality attribute but no utility fact, a utility fact is created setting the weight to the priority value of that preference fact and adding the preference fact to the superseded facts. |
| 5. Consolidate preferences | If there exists a utility fact and a non-superseded preference fact for a quality attribute, the utility fact's weight is updated and the preference fact is added to the list of superseded facts. |
| 6. Default authority level | If a stakeholder has a constraint for a quality attribute but does not have a specified authority level for it, a new authority fact is created setting the stakeholder's authority level to a default value. |

applicable as the reasoning engine for blackboard systems, e.g., in the domain of legal decision making [62]. Drools is based on the Rete algorithm [24] and supports both forward and backward chaining. In our case, we specified Drools rules in a dedicated file to handle conflict detection and resolution and insert new facts. Each rule is formulated based on a condition ("when") that triggers an action ("then"). Every rule can have a salience value indicating the priority of the rule, to ensure that if several rules fulfill their conditions, they will be fired in a deterministic order. An overview of the Drools rules for our blackboard system is shown in Table 3. It can be seen that the rules are concerned with superseded and conflicting constraints, concordance checks for preferences, the creation of new utility facts, the consolidation of preferences, and the creation of authority facts. These rules can be adjusted for future use cases, e.g., in case different resolution mechanisms are required for a certain application. For instance, for the concordance check (3. in Table 3), we check whether Kendall's concordance coefficient is at least 0.3 (see Step (D) in Sect. 3). Depending on the level of agreement required for a certain application, this value can be easily adjusted.

Listing 1 shows the rule for the detection of superseded lower bound constraints. In the example, a constraint fact $fact is superseded by another constraint fact $otherFact. It is superseded because $fact always holds when $otherFact is fulfilled. The rule's `when` condition requires a lower bound constraint fact $fact for a specific quality attribute that is not superseded, as well as another lower bound constraint fact $otherFact that has a constraint value greater than $fact's value. For instance, $fact might indicate that the speed of the robot should be at least 1 m/s, whereas $otherFact specifies that speed shall be at least 2 m/s. If the condition is fulfilled, the `then` part of the rule adds a message to the user indicating that $fact has been superseded and adds $fact to the collection of $otherFact's superseded facts.

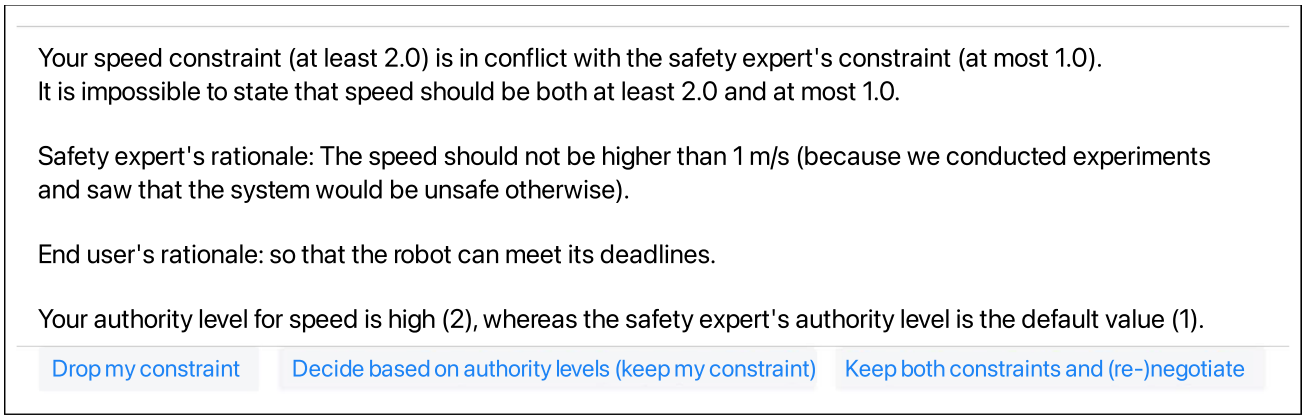


**Fig. 5** dialogue shown to the user to resolve a conflict related to two speed constraints

```
rule sumpersededLowerBound
salience 110
when
$fact: ConstraintFact($myQA: getQA(),
!isSuperseded(), isLowerBound(),
$myValue: getValue())

$otherFact: ConstraintFact(getQA() == $myQA,
isLowerBound(), getValue() > $myValue)
then
addMessage($otherFact + " supersedes " + $fact);
addToSuperseded($otherFact,$fact);
end
```

## 4.4 Conflict detection and resolution

Three mechanisms for conflict detection and resolution are supported: (1) Constraints can be in conflict with each other, (2) preferences can be non-concordant, or (3) preferences can be inconsistent. We describe these three cases in the following.

### 4.4.1 Conflicting constraints

As described in Step (D) in Sect. 3, conflicts between constraints can occur. Our tool supports conflict detection, explains conflicts to users, and suggests ways to resolve them. The supported options are to drop a constraint or to (temporarily) keep both and (re-)negotiate. Figure 5 shows a dialogue that is prompted to the end user describing an example conflict, stakeholders' rationales, authority levels, and options. Stakeholders' authority levels are crucial in situations in which no easy conflict resolution strategy can be found and it is necessary to decide between two conflicting constraints. In the example situation shown in Fig. 5, two speed constraints are in conflict with each other and the end user can decide which constraint should be kept, given that they have a higher authority level than the safety expert. The end user can decide to drop their constraint or drop the safety expert's constraint. If a constraint has been dropped, it is then possible to inspect them and access an explanation similar to the one in Fig. 4.

### 4.4.2 Non-concordant preferences

When preferences are not concordant, an information message by the consolidation agent is shown, explaining the issue of non-concordance and potential ways to solve the conflict. As described in Step (D) in Sect. 3, non-concordant preferences arise because the rankings of different stakeholders do not agree. To analyze what changes are required to create a concordant solution, the consolidation agent analyzes stakeholders' rankings and calculates possible changes to reach an agreement. In the current implementation, unilateral changes are considered, i.e., we analyze how an individual's ranking could be changed to reach a concordant solution. For instance, if a stakeholder has a first-ranked quality attribute that is not one of the other stakeholders' first-ranked quality attributes, we analyze which sliders/pairwise comparisons need to be adjusted to ensure that the second-ranked quality attribute has the same priority as the first-ranked quality attribute. Using an adjusted matrix, we run the AHP and calculate whether this change is sufficient to reach concordance.

> To reach a consensus, you need to align your preferences.
>
> - Option 1) @End user: To reach a concordant solution, it is enough if you lower the top slider and indicate that you strongly prefer speed over safety. If you do that, you slightly increase your ranking of safety, which is more in line with the others' preferences.

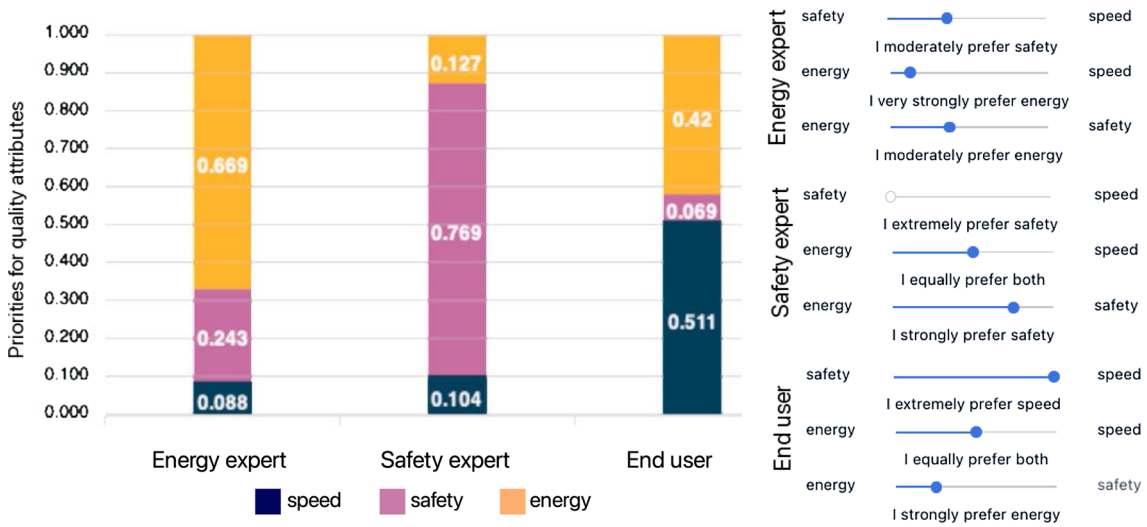


**Fig. 6** An overview of the stakeholders' preferences in our example

- Option 2) You can also convince the safety expert to lower their preference for safety. If the safety expert prefers safety as much as energy or speed, your preferences are concordant.
- Option 3) You can also convince the energy expert to lower their preference for energy. If the energy expert prefers energy as much as safety or speed, your preferences are concordant.

Write in the chat and negotiate with other stakeholders.

The stakeholders can then negotiate with each other using the chat.

#### 4.4.3 Inconsistent preferences

In case the AHP input of a single stakeholder is inconsistent (see Step (B) in Sect. 3), the web application provides an information message in the chat/log window. Inconsistencies can arise because pairwise comparisons are not proportional to each other or because they violate the property of transitivity. The following quote shows an example of what the information message can look like:

> Consolidation Agent: Your ranking is inconsistent.
> Your preferences indicate that
> (1) safety >> speed, (2) speed > energy, and (3) energy >> safety
> However, if you state that safety >> speed and speed > energy, it must follow that safety > energy.
> This is in conflict with your statement (3) (energy >> safety), which is why your ranking is inconsistent.

## 5 Example

The example system we use in the think-aloud study is a robot (e.g., a vacuum cleaner). The relevant quality attributes of the system are energy, safety, and speed. In this context, energy is measured by the battery charge of the robot and safety based on the expected number of collisions. Speed is measured in meters per second and thus related to the duration of the robot's mission. The stakeholder roles are energy expert, safety expert, and end user. In our study, the end user role was adopted by the participants. As mentioned before, stakeholder authority levels can be defined to indicate that a stakeholder is particularly knowledgeable when it comes to a certain quality attribute. In this example, a high stakeholder authority level is assigned to the energy expert for energy, the safety expert for safety, and the end user for speed. All other authority levels are set to a default value.

Figure 6 presents an overview of the stakeholders' preferences in our running example. The left part of the figure shows the bar chart depicting the stakeholders' priorities for the three quality attributes. They were calculated using AHP based on the user input shown in the right part of the figure. The example was designed in a way that stakeholders' priorities are not initially concordant: the energy expert clearly prioritizes energy, the safety expert has a strong preference for safety, and the end user prefers speed over the other quality attributes.

Apart from the preferences, the following constraints are specified:

1. Energy expert: Energy (battery charge) at least 5.0 — Rationale: The battery charge needs to be at least 5mAh so that the robot never runs out of energy

2. Safety expert: Speed at most 1.0 — Rationale: The speed should not be higher than 1 m/s (because we conducted experiments and saw that the system would be unsafe otherwise).
3. Safety expert: Safety (expected collisions) at most 2.5 — Rationale: We cannot accept more than 2.5 collisions because of SAFETYLEG363.
4. End user: Speed at least 2 — Rationale: The system needs to have a speed of at least 2 m/s (so that it can meet deadlines).
5. End user: Energy (battery charge) at least 1.0 — Rationale: The battery charge should be at least 1 (because it would be undesirable to run out of power).

It can be seen that the end user's speed constraint (5. in the list above) is in conflict with the safety expert's speed constraint (2.). In our study, the participants were presented with the issues of non-concordant preferences and constraint conflicts and asked to select appropriate resolution mechanisms.

## 6 Findings

To answer our research questions, we categorized our findings into themes focusing on the understandability of the tool (Sect. 6.1, RQ1), as well as on user satisfaction (Sect. 6.2, RQ2).

### 6.1 Understandability (RQ1)

When assessing our participants' mental models and understanding of the system, we identified that the overview that the system provided was very much appreciated. All participants had an immediate understanding of the preference bar charts and could read and interpret them without requiring any assistance. The list of constraints was also understandable, but required more processing time for our participants. Several participants stressed that they especially liked the rationales connected to constraints. When being asked about what the most helpful aspect of the tool was, a faculty member pointed out that it was "being able to drill down into this tool for some information." A researcher answered: "I see very clearly where everybody stands with respect to their position. I think the rationale was also useful, just seeing exactly why people say speed or safety is important."

#### 6.1.1 Preferences

When it comes to stakeholders' preferences, a majority of the participants found the sliders easy to use for pairwise comparisons. In our glitch detector task, all participants were able to validate whether a stakeholder's preference bar chart was in line with the values of the corresponding sliders or not.

At the same time, we found that it was difficult to judge whether different stakeholders' preferences were concordant without any additional tool explanations. When asking participants to perform prediction tasks using a set of stakeholders' preference bar charts, we found that our participants faced difficulties. A PhD student stated that "it is hard to say whether [the preferences] are concordant just by looking at them. I would have to write them down or analyze it more." Given a set of concordant preferences, several participants suggested that there might still be a discussion and indicated preferences that were not completely aligned. We found that it is not immediately apparent what changes are required to make preferences concordant. The explanations of non-concordance and the presentation of different options to reach an agreement were appreciated by the participants. A PhD student stated that it was helpful to get an overview of possible negotiation strategies to solve agreement issues:

> My favorite thing is how we get to the negotiation part at the end. It lists all of the possible changes that would make things work. Because it seems like that would be difficult to figure out if it didn't come straight out and tell you. By just looking at it, it's really hard to see if it's concordant or not.

A researcher pointed out that the transparent nature of the utility function definition process was beneficial:

> What I found helpful is the way the aggregated weight was being calculated. So, I give my own preferences. But then at the end without much work, I can quickly see that the weights are being found and then conflicts are being highlighted. And then conflicts are being attempted to resolve with an explanation. So all that gives me more transparency into what's going on and then I can think more about it.

An interviewee stressed that the explanations that guided stakeholders to create concordant preferences were especially useful. A faculty member suggested that even more explanations might be beneficial to help stakeholders understand the mechanisms of the tool, depending on the level of expertise of the users: "I think the system could show information or explanations on different levels of detail. One of the things I was wondering was how we actually go from the individual preferences to the final weights of the utility function."

#### 6.1.2 Constraints

The specification of constraints was considered "straightforward" by several participants. It was more challenging for

a few participants to reason about superseded and removed constraints. To analyze participants' understanding, we included several glitch detector tasks focusing on constraints that were superseded or removed. On average, it took our participants more than 30 seconds to identify glitches or arrive at the conclusion that the explanation was correct. A researcher stated that it was "tricky how one constraint was picked over another."

The current tool supports different mechanisms to remove constraints, either because a stakeholder decides to drop their constraint or because a stakeholder with a higher authority level decides to remove a constraint of a stakeholder with a lower authority level. It is also possible to start a conversation and agree on a new or modified constraint. When presented with these options, we found that our participants reasoned quite differently about constraint resolution. The constraints in question were speed constraints. The end user had a constraint requiring speed to be at least 2 m/s (because of deadlines), whereas the safety expert constrained speed to be at most 1 m/s. Given the same speed constraint-related conflict and adopting the end user role, we saw that the following decisions were taken by our participants:

1. Deciding that the own constraint shall be kept and removing the safety expert's constraint (because the end user had the top authority level for speed in our example)
2. Deciding to drop their constraint
3. Negotiating and convincing the other stakeholder to drop theirs

A majority of participants started a negotiation process in the chat. The participants deciding to go for decision 1.) insist on the fact that the top authority level was assigned to them (as the end users). After asking the safety expert about their rationales, one participant stated that "Well, as the end user, my main concern is meeting these deadlines. So I'm going to keep my constraints. And take a little [safety] risk." Another participant explained their decision as follows: "For me, dealing with end users is just that they are stubborn and obstinate, so I also went into that role, especially when dealing with deadlines. I know that you know a lot of people are very strict and concerned about meeting deadlines, and they panic about it, and they're willing to sacrifice safety." The participants who decided to drop their own constraints argued that the safety expert is an expert and would not add a speed constraint without having good reasons. One participant initially dropped the end user's constraint (speed at least 2 m/s) and added a new one (setting the speed to at least 1 m/s) that did not conflict with the safety expert's constraint.

One participant suggested that removing conflicting constraints should not be a suggested alternative.

> I feel like just dropping a constraint entirely might not make experts happy and they may be very unhappy and they may complain or walk away from it. So maybe there is a negotiation process that involves compromises and relaxation instead of overriding someone else's constraints.

A researcher stressed that the alternatives shown for constraint resolution were beneficial for the blackboard system's understandability.

> It was extremely clear what was going on. 'You said this, that person said that, here are some of the alternatives'. [...] and I can look at those alternatives and say: 'OK, I can live with alternative X' or I could say 'no there's no way to make those work for me' so then you keep going.

One participant stated that the constraint resolution "is what the tool is really good at. [...] And for humans it's hard to see when you have a lot of constraints [...] if they are satisfiable or not." For this reason, the participant considered the constraint consolidation aspect especially helpful in terms of understandability.

## 6.2 Satisfaction (RQ2)

To analyze how satisfied users are with the blackboard system's output, we collected Likert-scale answers measuring satisfaction levels. Our questions were inspired by the candidate Likert items used to evaluate user experience [23]. An overview of the answers is shown in Fig. 7. It can be seen that our participants indicated that they were generally satisfied with how information was considered and resolved. The usability of the system was not considered as positive as many of the other aspects. We asked our participants to motivate their answers and describe the findings below.

### 6.2.1 Negotiation

One finding related to negotiation was that different negotiation outcomes were considered satisfactory. Several participants aimed to ensure that their own concerns and preferences were well represented in the final utility function and constraints. Four participants actively searched for information about the rationales for constraints and preferences. When dealing with conflicting constraints, a staff member argued that they would use the negotiation part to make an informed decision: "I would want to understand why we're so far apart and what has led me to believe that a minimum speed of ten is required to complete the task. What has led them to believe that a maximum speed of nine is safe?"

The size of a required change was also an aspect that several participants took into account. One participant stressed

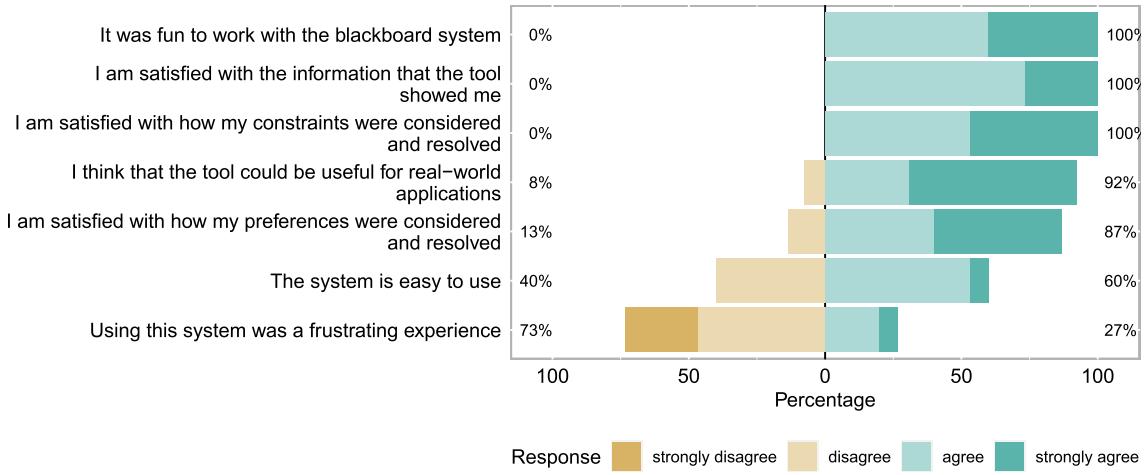


**Fig. 7** Likert-scale answers measuring the satisfaction levels ($n = 14$)

that avoiding unilateral changes and motivating various stakeholders to slightly cede their preferences could be beneficial: "It could be a smaller change for each of them than it would be if just one person decided to make a change. [...] But then you have to convince more people to change, which I imagine in practice could be harder than just convincing one person." When analyzing the alternatives to reach a consensus with respect to the preferences, another participant stated that it was acceptable to perform a small change on their own preferences, rather than starting a discussion with another stakeholder and motivating the other stakeholder to perform an even bigger change. One participant stressed that in certain situations, it might be easier to see that changes are necessary: "I might not want to change my preference [initially] but if I see that we're close to conversion, I might say that it's ok to change my preferences or drop a constraint."

#### 6.2.2 Real-world applications and scalability

Several participants mentioned that scalability could be an issue when deploying the tool for real-world applications. The issue that for many quality attributes, the AHP requires a large number of pairwise comparisons, was mentioned. When it comes to real-world applications, a faculty member asked how many discussions/negotiation conversations would typically occur. This interviewee suggested structuring the chat/log window in a better way and grouping conversations related to different conflicts. The faculty member also wondered whether there would always be pairwise negotiations between stakeholders or whether more than two stakeholders might be important when resolving conflicts. Another participant suggested that sending text messages in a chat is not the ideal way of exchanging information. Having a phone or video call could convey information more efficiently and effectively.

92% of the participants thought that the tool could be useful for real-world applications. One participant stressed that in practice, utility functions are often defined on an ad-hoc basis: "With the tool, I can actually reason about the utility function. Otherwise, I would simply create a function out of the blue and not put in a lot of thought."

One participant pointed out that this tool is especially beneficial because it could be used early on in the requirements engineering process and not when all components of the system are already built. Another participant stressed that the tool might be especially useful when a running system is analyzed and the preferences are re-adjusted. This participant stated that data from simulations or the running system would be beneficial to understand the impact of the utility function's weights, so that "you run the system, you see how it behaves and then you can adjust the utility function that way, too."

#### 6.2.3 Usability

The usability of the tool was one of the areas where our participants had suggestions for improvement. 40% of the participants indicated that the system is not easy to use and 27% found it frustrating to work with the system. Several interviewees stated that it would be difficult to work with the tool on their own, without guidance from a tutorial or tool expert. The setup of the study as a think-aloud session was considered beneficial, as it allowed for clarification and support where needed.

The visualizations were stressed as very helpful by a majority of the participants. A PhD student stated:

> "I particularly liked the bar plots of the utility of each attribute for each stakeholder. The discussion/chat can also be useful. I like how it is easy to specify your

preferences between attributes. I think it's intuitive to just say 'I prefer this one over that one."'

The number lines that visualize the upper/lower bounds of superseded/conflicting constraints (see Fig. 4) were considered more difficult to understand, but were still considered helpful by multiple participants.

## 7 Related work

Utility functions have been widely applied in the context of human decision-making, in particular, based on von Neumann and Morgenstern's contributions to expected utility theory [66]. In the field of optimization in autonomic computing systems, utility functions have become widely used since the early 2000s [70]. While utility functions are a common mechanism in self-adaptive systems [12, 16, 22, 26, 31, 61], there exist only a few approaches to defining them. This paper addresses the need for preference elicitation techniques to ensure that utility functions meet stakeholders' needs [70]. In the following, we describe several related approaches.

Adjusting utility functions at run time: It is important to keep in mind that utility functions cannot be specified once and for all at design time, but that elicitation and readjustment of preferences at run time is typically needed, especially when it comes to self-adaptive systems [39]. Song et al. [60] propose an approach that collects user feedback after every round of adaptation to adjust the weights of constraints. A related approach relies on user feedback to switch between "variants" with associated utility function weights, depending on the current usage context [36].

When prioritizing quality attributes at run time, it is often crucial to consider the system's context and adapt requirements [17, 40, 59]. One approach [59] uses the AHP for pairwise comparisons of quality attributes while taking contextual factors into consideration (e.g., related to the urgency of tasks, the time period, or weather). The authors concluded that the proposed elicitation technique was beneficial, but that it is difficult to avoid overwhelming users when eliciting preferences along with a large amount of contextual information and scenarios. User-adaptive task models are used by another approach that captures users' tasks, contextual factors, as well as preferences [58]. Based on this information, the self-adaptive system's behavior can be adapted whenever user preferences are readjusted, the context changes, or failures occur [58].

Several related techniques have been developed that explicitly take the uncertainty of self-adaptive systems' contexts into account. The ARRoW approach uses Primitive Cognitive Network Process (P-CNP), an improved version of the AHP, and dynamic decision networks to reassess utility weights at run time [48]. Partially Observable Markov Decision Processes are used by another approach to model the satisficement of non-functional requirements, explicitly considering the uncertainty of run-time contexts [47]. We acknowledge the need to support varying system contexts and consider it promising to extend our approach with automatic mechanisms to adjust utility functions at run time. We envision our approach to be used continuously, so that utility function weights can be adjusted based on preference elicitation and a consensus between multiple stakeholders even when the system is running. Quantifying contributions in goal models: Besides using utility functions, goal-oriented approaches are also a common mechanism to capture system objectives in self-adaptive systems [70]. In the context of goal-oriented requirements engineering, contribution labels are commonly used to indicate how much goals contribute to each other's satisficement [33, 44]. These contribution labels can be qualitative (e.g., "–" or "+") or quantitative (e.g., 0.8 or 0.1). Although it has been criticized that quantitative labels add unwarranted precision and can overwhelm users, they have still been found to be beneficial in empirical studies [44]. Several approaches to quantifying the contributions of goals have been developed and many of them are based on similar techniques as ours. For instance, an approach for self-adaptive systems uses goal models that can be analyzed, converted into arithmetic functions, and leveraged to select optimal adaptation strategies at run time [4]. It is suggested to use group decision techniques and AHP to arrive at the weights of goals' contributions. A similar study has also successfully combined AHP for the quantification of goal contributions with group decision techniques [3]. Our approach does not require stakeholders to create a complete model of goals, actors, and their relationships, but focuses on multi-stakeholder preference elicitation to create a utility function encoding the key quality attributes.

Utility functions in the context of goal models can also be used to determine how multiple functional requirements contribute to the satisficement of non-functional requirements in self-adaptive systems. For instance, Providentia [10] uses a search-based technique to determine the weights of such utility functions at run time with the goal of maximizing overall satisficement of requirements. The proposed approach was found to lead to better and more robust results than setting the weights manually or randomly [10].

Analytic Hierarchy Process for requirements prioritization: The AHP has been used for analyzing requirements trade-offs, especially because of its favorable mathematical properties (e.g., consistency and concordance checks) [18, 45, 55]. One of the known disadvantages is the large number of required comparisons, since $n(n-1)/2$ comparisons need to be performed for $n$ quality attributes. In practice, it can be difficult for end users to assign absolute values for the pairwise comparisons [37, 67]. In certain

situations, it can be sufficient to use an ordinal scale, rather than relying on ratio scale data (as in the case of the AHP) [37]. Future extensions of our work can explore other prioritization techniques, especially for applications where a large number of quality attributes needs to be considered. To deal with uncertainty and the difficulty of selecting precise values when comparing quality attributes, fuzzy extensions of AHP have been proposed [41].

Requirements negotiation and conflict resolution: Several requirements negotiation techniques have been proposed in the last decades [8, 28, 29, 57]. The WinWin spiral model [8] is an early, well-known negotiation approach that helps multiple stakeholders gain an understanding of their conflicts and arrive at a mutual agreement. It has led to the development of other negotiation approaches, e.g., EasyWinWin [28], an approach that is based on a group support system for negotiation and conflict resolution. Many of the existing negotiation techniques require an analysis of stakeholders, their objectives, and potential conflicts before actual negotiation starts. The conflict detection mechanisms in our negotiation support system address this need and can assist stakeholders in identifying conflicts (semi-)automatically. Different conflict resolution strategies might be beneficial in different situations. Tools to automatically detect and resolve requirements conflicts have been developed in the past, especially in the context of goal-driven requirements engineering [51, 64]. For instance, Oz is a tool that can automatically detect conflicts, categorize them, and generate compromise resolutions using planning techniques [52]. Typically, multiple resolution alternatives exist and human input can be leveraged to decide how a conflict should be resolved in a specific situation. We support a subset of resolution alternatives in our tool (Sect. 4.4) and found that our participants adopted different conflict behavior, which is in line with previous findings [29, 63].

Requirements prioritization is strongly connected to requirements negotiation and often used as an input to focus the negotiation process [6]. Our approach is based on these insights and leverages AHP as a prioritization technique in an initial step to inform the negotiation process. To support distributed settings, some of the activities in our method are performed individually by each stakeholder, whereas for the actual negotiation, we recommend participants to collaborate synchronously using the chat. This recommendation is in line with the state of the art of requirements negotiation tools, in which both synchronous and asynchronous collaboration are supported [29].

## 8 Discussion and future work

Our think-aloud study indicated that the explanations provided by the tool, as well as the conflict resolution mechanisms, helped to establish an understandable and transparent utility function definition process with traceability to the initial input. Participants were generally able to identify glitches in explanations and required considerable time effort when aiming to identify and resolve conflicts on their own. Our interview data suggest that they were generally satisfied with the tool support, although refinements to the usability are needed to increase the maturity of the tool further. What should be noted is that the large number of elements that were included in our dashboard resulted in the user interface being perceived as crowded, especially on devices with smaller screens. Redesigning the interface by introducing further navigation elements is one of the areas of future work.

An important aspect to consider is the level of abstraction at which quality attributes shall be compared and reasoned about. Our approach assumes that involved stakeholders are aware of the quality attributes under consideration and know how they can be measured. For our weighted sum approach for utility function definition, quantifiable quality attributes are required. For instance, the participants in our study were informed that we considered the expected number of collisions when reasoning about safety. These aspects need to be taken into account when performing pairwise comparisons of quality attributes (and possibly even details of how the system is or will be implemented). To acknowledge the need for other forms of requirements elicitation, we also support the collection and specification of constraints in our approach. We expect the negotiation support system to be adjustable to different kinds of utility functions and input (e.g., requirements at lower levels of abstraction) that can be prioritized, consolidated, and reasoned about in a collaborative effort.

We decided to focus on the weighted sum approach for utility function definition in this paper, given that it is applied in actual systems (e.g., [12, 21, 26, 61]) and we aim to address a real-world concern with this research. It should be noted that the weighted sum approach has the property that quality attribute dimensions can be traded off against each other—poor performance in one dimension can be compensated by good performance in another dimension. In certain situations, it would be more beneficial to define nonlinear utility functions. For instance, multiplying the utilities of different quality attributes can allow stakeholders to express logical “and”s and capture a conjunction of constraints. Independently of whether a utility function is described as a weighted sum or not, real-world contexts commonly require eliciting the priorities of requirements and negotiating constraints [6, 29]. We are convinced that our proposed approach is of value to other requirements prioritization and negotiation contexts and not only relevant to define weighted sum utility functions.

We explicitly focus on self-adaptive systems in this paper; however, the developed negotiation support system might

be applicable to other contexts as well. The focus on self-adaptive systems is motivated by the fact that many existing self-adaptive systems rely on utility functions (e.g., [12, 21, 22, 26, 31, 35, 56, 61]) and we aimed to focus on real-world problems in our research. Future work will investigate the applicability of our approach to other contexts in which requirements negotiation and conflict resolution are needed. Non-self-adaptive systems generally have different kinds of requirements that are not necessarily expressed in utility functions, but whose consolidation would lead to different strategic or design decisions [7].

Our blackboard system's architecture supports future extension and customization of our approach. To support extension, we aimed to design the tool with a focus on the separation of concerns and pluggability of context-specific elements. For instance, we use a variety of "fact types" (e.g., constraint facts, preference facts, and definition facts) and developed different kinds of agents to insert and process facts. Resolution policies can easily be adjusted by changing the consolidation agent's Drools rules (Sect. 4.3). It is also possible to add further agents that are not limited to our Drools-based agent and the Vaadin-based user interface. For instance, the current tool can be augmented with analysis agents relying on run-time or simulation data, as also suggested by two participants. These mechanisms would allow users to analyze and see the impact of their preferences and utility function weights on system behavior. Such a tool could generate different plans to show users how the actual behavior of the system would be affected by changes to users' preferences or to the utility function. For example, it could be stated that a different path would be selected if a user changed their preference for a specific pair of quality attributes in a certain way. Such analyses could help stakeholders to determine what their actual preferences are and whether a utility function meets their needs.

Further future extensions of the tool include adding support for other roles that partially reuse existing agents' functionality (e.g., legal experts specifying different kinds of hard requirements or human facilitators that do not specify any own preferences but support the negotiation phase). The negotiation support system can be adjusted to process different kinds of information, e.g., hard or soft constraints, goals, other kinds of utility functions, scenario-specific information, stakeholder roles, or quality attributes. We imagine the system to be used continuously, so that stakeholders can engage in a discussion even as the system context or environment changes. Mechanisms to allow users not only to express their preferences on a general level, but elicit situation-specific preferences and utility functions, are another area for future work.

An interesting observation of our study is that it confirms previous findings related to requirements negotiation. The negotiation support system helps with the automatic identification of conflicts and the proposition of alternative solutions, which are two of the crucial activities in the requirements negotiation process [2]. Moreover, different conflict behaviors reported by Thomas [63] were observable in our study: Given the same role description of an end user, some participants adopted a competing role, whereas others were accommodating, and others used compromising conflict resolution strategies. It is important to keep in mind that humans react differently when facing conflicts. Future approaches can build upon these lessons and ensure that negotiation dynamics are not deteriorated by too competing stakeholders and that crucial stakeholder input is still elicited. It should be noted that our participants were provided with a role description rather than representing their own opinions. We expect stakeholders in real-world situations to engage more strongly and insist more heavily on their positions than in our think-aloud study. The negotiation dynamics in real-world situations are likely to be different from the ones in our think-aloud study. For instance, we expect that future case studies discover different negotiation tactics and ways of reasoning than the ones reported in Sect. 6.2.1. In practice, personal relations between stakeholders and an in-depth understanding of the constraints' rationales in their real-world contexts certainly have an impact on negotiation. While the findings we describe in the paper indicate how humans reason about negotiation in general, we acknowledge the need for a case study to explore the phenomenon of requirements negotiation and prioritization in a practical setting.

## 9 Summary and conclusions

This paper presented a method that supports multiple stakeholders in eliciting constraints, prioritizing relevant quality attributes, negotiating, and giving input to define utility functions for self-adaptive systems. The tool-supported method is based on the AHP for the pairwise comparison of quality attributes and is supported by a blackboard system that centrally stores information and coordinates several agents. We implemented a consolidation agent that uses the reasoning engine Drools to process information, identify conflicts, and suggest resolution mechanisms to help stakeholders arrive at a utility function.

To assess the approach with respect to its understandability and user satisfaction, we performed a think-aloud study with 14 participants. Our study sheds light on how differently humans reason about and how they negotiate around quality attributes. We found that it can be difficult for participants to manually identify conflicts and arrive at concordant preferences. Our tool's mechanisms for conflict detection, (semi-)automatic conflict resolution, and visualization of preferences were perceived as very useful. Overall,

our approach helps to make the process of utility function definition more understandable and transparent.

The developed method and tool support appear useful and applicable to other domains and systems that could benefit from requirements negotiation. A promising direction for future work is to perform case studies to investigate our approach's applicability in practical contexts. Moreover, future work can build upon the blackboard system and add other kinds of information/requirements that are of relevance to utility function definition. For instance, contextual information is important to consider for real-world self-adaptive systems. This information can be complemented with support for analysis tools that simulate and explain the impact of different utility functions on the behavior of the system.

**Acknowledgements** We would like to thank all participants for their help and support with the study. This work is supported in part by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by award N00014172899 from the Office of Naval Research and by the NSA under Award No. H9823018D000. Any views, opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Office of Naval Research or the NSA.

## References

1. Abdennadher I, Rodriguez IB, Jmaiel M (2018) A utility-based approach for self-adaptive systems: Application to a smart building. In: Proceedings of the IEEE/ACS 14th international conference on computer systems and applications (AICCSA), pp 76–82. https://doi.org/10.1109/AICCSA.2017.41
2. Ahmad S (2008) Negotiation in the requirements elicitation and analysis process. In: Proceedings of the 19th Australian conference on software engineering (ASWEC 2008), pp 683–689. https://doi.org/10.1109/ASWEC.2008.4483263
3. Akhigbe O, Alhaj M, Amyot D, Badreddin O, Braun E, Cartwright N, Richards G, Mussbacher G (2014) Creating quantitative goal models: governmental experience. In: Yu E, Dobbie G, Jarke M, Purao S (eds) Conceptual modeling. Springer International Publishing, Cham, pp 466–473
4. Anda AA (2020) Combining goals and SysML for traceability and decision-making in the development of adaptive socio-cyber-physical systems. Ph.D. thesis, Université d'Ottawa/University of Ottawa
5. Asadi M, Soltani S, Gasevic D, Hatala M, Bagheri E (2014) Toward automated feature model configuration with optimizing non-functional requirements. Inf Softw Technol 56(9):1144–1165. https://doi.org/10.1016/j.infsof.2014.03.005
6. Berander P, Andrews A (2005) Requirements prioritization. In: Engineering and managing software requirements, pp 69–94. Springer
7. Boehm B (2003) Value-based software engineering: reinventing. ACM SIGSOFT Softw Eng Notes 28(2):3
8. Boehm B, Bose P, Horowitz E, Lee MJ (1994) Software requirements as negotiated win conditions. In: Proceedings of the international conference on requirements engineering, May 1994, pp 74–83. https://doi.org/10.1109/icre.1994.292400
9. Bowers KM, Fredericks EM, Cheng BHC (2018) Automated optimization of weighted non-functional objectives in self-adaptive systems. In: Colanzi TE, McMinn P (eds) Search Based Softw Eng. Springer International Publishing, Cham, pp 182–197
10. Bowers KM, Fredericks EM, Hariri RH, H C Cheng B (2020) Providentia: Using search-based heuristics to optimize satisficement and competing concerns between functional and non-functional objectives in self-adaptive systems. J Syst Softw 162, 110497. https://doi.org/10.1016/j.jss.2019.110497
11. Cegan JC, Filion AM, Keisler JM, Linkov I (2017) Trends and applications of multi-criteria decision analysis in environmental sciences: literature review. Environ Syst Dec 37(2):123–133
12. Cheng SW, Garlan D, Schmerl B (2006) Architecture-based self-adaptation in the presence of multiple objectives. In: Proceedings of the 2006 international workshop on self-adaptation and self-managing systems. https://doi.org/10.1145/1137677.1137679
13. Corkill DD (1991) Blackboard systems. AI Exp 6:40–47
14. Creswell JW (2008) Research design: qualitative, quantitative, and mixed methods approaches, 3 edn. Sage Publications Ltd.
15. Curtain C (2021) QualCoder 2.4 [Computer software]. https://github.com/ccbogel/QualCoder/releases/tag/2.4
16. Cámara J, Lopes A, Garlan D, Schmerl B (2016) Adaptation impact and environment models for architecture-based self-adaptive systems. Sci Comput Program 127:50–75. https://doi.org/10.1016/j.scico.2015.12.006
17. Dell'Anna D, Dalpiaz F, Dastani M (2019) Requirements-driven evolution of sociotechnical systems via probabilistic reasoning and hill climbing. Auto Softw Eng 26(3):513–557
18. Elahi G, Yu E (2012) Comparing alternatives for analyzing requirements trade-offs—in the absence of numerical data. Inf. Softw. Technol. 54(6):517–530. https://doi.org/10.1016/j.infsof.2011.10.007
19. Ericsson KA, Simon HA (1984) Protocol analysis: verbal reports as data. MIT Press
20. Erman LD, Hayes-Roth F, Lesser VR, Reddy DR (1980) The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty. ACM Comput Surv 12(2):213–253. https://doi.org/10.1145/356810.356816
21. Esfahani N, Elkhodary A, Malek S (2013) A learning-based framework for engineering feature-oriented self-adaptive software systems. IEEE Trans Softw Eng 39(11):1467–1493
22. Faniyi F, Lewis PR et al (2014) Architecting self-aware software systems. In: WICSA'14, pp 91–94
23. Finstad K (2010) The usability metric for user experience. Interact Comput 22(5):323–327
24. Forgy CL (1989) Rete: A fast algorithm for the many pattern/many object pattern match problem. Readings in Artif Intell Databases, pp 547–559. Elsevier
25. Galitz WO (2007) The Essential guide to user interface design: an introduction to GUI design principles and techniques. John Wiley & Sons Inc., New York
26. Ghezzi C, Molzam Sharifloo A (2013) Dealing with non-functional requirements for adaptive systems via dynamic software product-lines, pp 191–213. Springer, Berlin
27. Grönroos M (2011) Book of Vaadin. Vaadin.com
28. Grünbacher P (2000) Collaborative requirements negotiation with EasyWinWin. In: Proceedings 11th international workshop on database and expert systems applications, pp 954–958. IEEE
29. Grünbacher P, Seyff N (2005) Requirements negotiation. In: Engineering and managing software requirements, pp 143–162. Springer
30. Hauser JR, Urban GL (1979) Assessment of attribute importances and consumer utility functions: Von

neumann-morgenstern theory applied to consumer behavior. J Consum Res 5(4):251–262
31. Heaven W, Sykes D, Magee J, Kramer J (2009) A case study in goal-driven architectural adaptation. In: Software engineering for self-adaptive systems, p 109–127. Springer-Verlag, Berlin. https://doi.org/10.1007/978-3-642-02161-9_6
32. Hoffman R, Mueller S, Klein G, Litman J (2018) Metrics for explainable AI: challenges and prospects. XAI Metrics
33. Horkoff J, Yu E (2013) Comparison and evaluation of goal-oriented satisfaction analysis techniques. Require Eng 18(3):199–222. https://doi.org/10.1007/s00766-011-0143-y
34. Hsu CC, Sandford BA (2007) The Delphi technique: making sense of consensus. Pract Assess Res Eval 12(1):10
35. Inverardi P, Mori M (2013) A software lifecycle process to support consistent evolutions. In: R. de Lemos (ed.) Self-adaptive systems, vol 7475 LNCS, pp 239–264. Springer, Berlin
36. Kakousis K, Paspallis N, Papadopoulos G (2008) Optimizing the utility function-based self-adaptive behavior of context-aware systems using user feedback. In: OTM 2008, pp 657–674
37. Karlsson L, Host M, Regnell B (2006) Evaluating the practical use of different measurement scales in requirements prioritisation. Proceedings of the 5th ACM-IEEE international symposium on empirical software engineering 2006, 326–335 (2006). https://doi.org/10.1145/1159733.1159782
38. Kendall MG, Smith BB (1939) The problem of *m* rankings. Ann. Math. Statist. 10(3):275–287
39. Kephart J (2021) Viewing autonomic computing through the lens of embodied artificial intelligence: a self-debate
40. Knauss A, Damian D, Franch X, Rook A, Müller HA, Thomo A (2016) ACon: a learning-based approach to deal with uncertainty in contextual requirements at runtime. Inf Softw Technol 70:85–99. https://doi.org/10.1016/j.infsof.2015.10.001
41. Krejčí J (2018) Pairwise Comparison matrices and their Fuzzy extension. Springer
42. Lethbridge TC, Sim SE, Singer J (2005) Studying software engineers: data collection techniques for software field studies. Empirical Softw Eng 10(3):311–341. https://doi.org/10.1007/s10664-005-1290-x
43. Lewis C (1982) Using the "thinking-aloud" method in cognitive interface design. IBM TJ Watson Research Center Yorktown Heights, NY
44. Liaskos S, Hamidi S, Jalman R (2013) Qualitative vs. quantitative contribution labels in goal models: Setting an experimental agenda. In: Proceedings of the 6th international i* workshop (iStar 2013), iStar, pp 37–42
45. Liaskos S, Jalman R, Aranda J (2012) On eliciting contribution measures in goal models. In: Proceedings of the 20th IEEE international requirements engineering conference, pp 21–230. IEEE. https://doi.org/10.1109/RE.2012.6345808
46. Ossadnik W, Schinke S, Kaspar RH (2016) Group aggregation techniques for analytic hierarchy process and analytic network process: a comparative analysis. Group Dec Negot 25(2):421–457
47. Paucar LHG, Bencomo N (2018) RE-STORM: mapping the decision-making problem and non-functional requirements trade-off to partially observable markov decision processes. In: Proceedings of the 13th international conference on software engineering for adaptive and self-managing systems, SEAMS '18, pp 19–25. Association for Computing Machinery, New York, NY, USA . https://doi.org/10.1145/3194133.3195537
48. Paucar LHG, Bencomo N, Yuen KKF (2019) ARRoW: automatic runtime reappraisal of weights for self-adaptation. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing, pp 1584–1591
49. Poladian V, Sousa JP, Garlan D, Shaw M (2004) Dynamic configuration of resource-aware services. In: Proceedings of the 26th international conference on software engineering, pp 604–613 . https://doi.org/10.1109/ICSE.2004.1317482
50. Proctor M (2011) Drools: a rule engine for complex event processing. In: International symposium on applications of graph transformations with industrial relevance, pp 2–2. Springer
51. Robinson WN (1996) Automated assistance for conflict resolution in multiple perspective systems analysis and operation. In: Joint proceedings of the 2nd international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT'96 workshops, pp 197–201
52. Robinson WN, Fickas S (1994) Supporting multi-perspective requirements engineering. In: Proceedings of IEEE international conference on requirements engineering, pp 206–215. IEEE
53. Rojas JM, Fraser G, Arcuri A (2015) Automated unit test generation during software development: A controlled experiment and think-aloud observations. In: ACM international symposium on software testing and analysis (ISSTA), 2015. ACM
54. Saaty R (1987) The analytic hierarchy process–what it is and how it is used. Math Modell 9(3):161–176
55. Salehie M, Tahvildari L (2012) Towards a goal-driven approach to action selection in self-adaptive software. Softw Pract Exp **42**(2), 211–233
56. Sawyer P, Bencomo N, et al (2010) Requirements-aware systems: a research agenda for RE for self-adaptive systems. In: RE'10, pp 95–103
57. Schoop M, Jertila A, List T (2003) Negoisst: a negotiation support system for electronic business-to-business negotiations in e-commerce. Data Knowl Eng 47(3):371–401
58. Serral E, Sernani P, Dalpiaz F (2018) Personalized adaptation in pervasive systems via non-functional requirements. J Ambient Intell Human Comput 9(6):1729–1743
59. Serral E, Sernani P, Dragoni AF, Dalpiaz F (2017) Contextual requirements prioritization and its application to smart homes. In: Braun A, Wichert R, Maña A (eds) Ambient intelligence. Springer International Publishing, Cham, pp 94–109
60. Song H, Barrett S, Clarke A, Clarke S (2013) Self-adaptation with end-user preferences: using run-time models and constraint solving. In: MODELS'13
61. Sousa JP, Balan RK, Poladian V, Garlan D, Satyanarayanan M (2008) User guidance of resource-adaptive systems. In: ICSOFT 2008, pp 36–44
62. Szymanski L, Sniezynski B, Indurkhya B (2018) A multi-agent blackboard architecture for supporting legal decision-making. Comput Sci **19**(4)
63. Thomas KW (1992) Conflict and conflict management: Reflections and update. J Org Behav pp 265–274
64. Van Lamsweerde A, Darimont R, Letier E (1998) Managing conflicts in goal-driven requirements engineering. IEEE Trans Softw Eng 24(11):908–926. https://doi.org/10.1109/32.730542
65. Van Someren MW, Barnard YF, Sandberg JA (1994) The think aloud method: a practical approach to modelling cognitive processes, 1 edn. Academic Press
66. Von Neumann J, Morgenstern O (1953) Theory of games and economic behavior. Princeton University Press, Princeton
67. Voola P, Babu AV (2013) Comparison of requirements prioritization techniques employing different scales of measurement. ACM SIGSOFT Softw Eng Notes 38(4):1–10. https://doi.org/10.1145/2492248.2492278
68. Vora P (2009) Web application design patterns. Morgan Kaufmann
69. Wallace C, Cook C, Summet J, Burnett M (2002) Assertions in end-user software engineering: a think-aloud study. In:

Proceedings IEEE 2002 symposia on human centric computing languages and environments, pp 63–65. IEEE
70. Walsh WE, Tesauro G, Kephart JO, Das R (2004) Utility functions in autonomic systems. In: Proceedings of the international conference on autonomic computing, May 2014, pp 70–77. https://doi.org/10.1109/ICAC.2004.1301349
71. Wohlrab R, Garlan D (2021) Defining utility functions for multi-stakeholder self-adaptive systems. In: F. Dalpiaz, P. Spoletini (eds.) Require Eng Found Softw Quality, pp 116–122. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-73128-1_8