# Using the Crowds to Satisfy Unbounded Requirements

Fabiano Dalpiaz and Michal Korenko
Utrecht University, the Netherlands
f.dalpiaz@uu.nl, m.korenko@students.uu.nl

Rick Salay and Marsha Chechik
University of Toronto, Canada
{rsalay, chechik}@cs.toronto.edu

***Abstract*—The Internet is a social space that is shaped by humans through the development of websites, the release of web services, the collaborative creation of encyclopedias and forums, the exchange of information through social networks, the provision of work through crowdsourcing platforms, etc. This landscape offers novel possibilities for software systems to satisfy their requirements, e.g., by retrieving and aggregating the information from Internet websites as well as by crowdsourcing the execution of certain functions. In this paper, we present a special type of functional requirements (called *unbounded*) that is not fully satisfiable and whose satisfaction is increased by gathering evidence from multiple sources. In addition to characterizing unbounded requirements, we explain how to maximize their satisfaction by asking and by combining opinions of multiple sources: people, services, information, and algorithms. We provide evidence of the existence of these requirements through examples by studying a modern Web application (Spotify) and from a traditional system (Microsoft Word).**

## I. Introduction

The modern Internet goes beyond its conception as a network of computers [1]. In fact, it is a *social space* [2] shaped by the actions of and the interactions among humans, i.e., by the development of websites, the release of web services, the collaborative creation of encyclopedias and forums, the exchange of information through social networks, the offering and provision of work through crowdsourcing platforms, etc.

This landscape offers novel, unprecedented possibilities for the satisfaction of many requirements of software systems by orchestrating the heterogeneous sources that exist in the Internet, as opposed to relying on the traditional algorithm and database model [3]. For example, grammar checking is a notoriously difficult problem for an algorithm[1] that people can solve with higher accuracy. To such extent, a software system could make use of people by posting a grammar checking job on the Amazon Mechanical Turk crowdwork platform[2].

In principle, many types of requirements can be addressed by exploiting sources from the Internet: *information* available in websites, *services* that can be invoked through web interfaces, and *people* that can be consulted through social media, e-mail, crowdwork platfoms, and instant messaging.

In this paper, we focus on a special type of requirement that inherently calls for integration with the Internet to be optimally satisfied: *unbounded requirements*. These are functional requirements that are impossible or impractical to satisfy fully, and their satisfaction is increased by gathering evidence. These requirements are relatively common in practice (see Sec. IV), and there are different reasons for their unboundedness.

For example, consider two requirements for a hypothetical event planning system: $R_1$: "Enumerate *suitable* participants for an event", and $R_2$: "Do not allow a *spam* event to be scheduled". Both are unbounded: they are functional, and one can hardly envision how they can be fully satisfied. Their unboundedness differs though: $R_1$ requires (human) intelligence to determine suitability, and different answers—lists of participant names—are acceptable, as long as proper argument for inclusion are provided; $R_2$ is ungeneralizable: whether an event is *spam* is binary, but it is impossible to define a fixed, finite set of rules to check this.

As we show in the paper, the satisfaction of unbounded requirements can be increased by consulting multiple sources both within and outside the system's context and by combining their inputs. For example, for $R_2$, one could concurrently run an algorithm with a fixed set of rules, ask expert event organizers to give an opinion, and post a task for checking this on a crowdwork platform like Amazon Mechanical Turk.

Specifically, we make the following contributions:

- We characterize unbounded requirements through a framework (a definition and set of unboundedness types) that can be used to identify these requirements;
- We explain how these requirements can be optimally satisfying by combining multiple sources from the Internet;
- We report on their relevance, by discussing concrete examples from Microsoft Word and Spotify.

The rest of this paper is organized as follows. In Sec. II, we introduce unbounded requirements. In Sec. III, we discuss strategies for maximizing their satisfaction. In Sec. IV, we describe examples from two software applications. We conclude in Sec. V with a summary and future research directions.

## II. Unbounded Requirements

This section introduces unbounded requirements. After providing a framework for their identification in Sec. II-A, we contrast them to other requirement types in Sec. II-B.

### A. Characterization

The key characteristic of unbounded requirements is an inherent unbounded quality making it impossible to devise

[1] See the authoritative viewpoint of Geoffrey Pullum on algorithmic grammar checkers: http://itre.cis.upenn.edu/ myl/languagelog/archives/005061.html

[2] https://www.mturk.com/

TABLE I
EXAMPLES OF COMMON CASES FOR THE UNBOUNDEDNESS OF A REQUIREMENT

| Case | Example requirement | Why unbounded |
|---|---|---|
| **Intelligent** | Summarize an event's description. | This requires human intelligence, for generating a summary is more than shortening a long description, and requires skills that are hardly merely rational. |
| **Subjective** | Determine an appropriate venue for an event. | The determination of appropriateness is inherently subjective and the answer can always be improved by collecting more opinions and relevant data (e.g., historical, statistical, etc.). |
| **Ungeneralizable** | Check the grammar of an event description. | It is generally accepted that grammar checking has so many exceptions that a corpus-based approach is more effective than a fixed algorithm based on a set of grammar rules. |
| **Dynamic** | Check the spelling of a word. | Language is constantly evolving so any fixed dictionary is quickly likely to be incorrect. In addition, the set of proper names is unbounded. |
| **Uncertain** | Determine if weather will adversely affect an event. | Weather has inherently chaotic characteristics that make it difficult to predict. |
| **Impractical** | Determine how many people in a city like a specific venue. | To fully satisfy this we would need to query every person in the city but in practice this may be too expensive. An alternative is to approximate it (i.e., partially satisfy it) by gathering evidence such as through the use of demographic information coupled with venue rating services. |

a specification that will fully satisfy them. This intuition is captured by the following definition.

*Definition 1 (Unbounded Requirement):* An *unbounded requirement* is a functional requirement that can only be partially satisfied, and the level of satisfaction can be increased by gathering evidence for its satisfaction.

Def. 1 includes requirements that are "practically" unbounded: while in principle they can be fully satisfied, in practice they cannot. In this paper, we focus on unbounded *query* requirements that are satisfied by collecting or computing some data that is an answer to a query. This is contrasted with unbounded *action* requirements that are satisfied by performing an action that changes a state of the world. For example, "the system must track the current temperature" is a query requirement while "the system must reconcile the bank accounts" is an action requirement. We can improve the satisfaction of an unbounded query requirement by *gathering evidence for different possible answers to a query*. The study of unbounded action requirements is left for future work.

We identify a non-exhaustive list of common cases of unbounded query requirements. A requirement may belong to more than one case. Table I exemplifies each case.

- **Intelligent**: Satisfying these requirements needs human judgment, understanding or creativity. Often these are requirements whose satisfaction is dependent on producing a sufficiently convincing argument. These are only partially satisfiable because there are many possible "right" answers, although some may be better than others.
- **Subjective**: They require to determine the answer to a query that is inherently subjective. These are only partially satisfiable because more evidence for/against a subjective answer can always be obtained to improve it.
- **Ungeneralizable**: They have many input/output examples or cases but cannot be fully generalized into a fixed set of rules. These are only partially satisfiable because, although they may be satisfiable for some inputs, they cannot be satisfied for all inputs.
- **Dynamic**: They can be fully satisfied in a narrow time window but not over a large period, because they rely on conditions that are constantly changing.
- **Uncertain**: Their satisfaction needs the ability to predict something that is inherently uncertain. These are only partially satisfiable because the prediction is not guaranteed to always be correct.
- **Impractical**: They can be fully satisfied in theory but in practice it is too costly or time-consuming to do so. Thus, they are partially satisfied using approximations.

### B. Relation with Traditional Requirements Types

In this section, we contrast unbounded requirements with the most relevant categories of requirements from the literature. As Def. 1 states, unbounded requirements are a special case of functional requirements (as they define services that the system should provide) that cannot be fully satisfied and their satisfaction is increased by gathering evidence.

Despite the fact that they belong in the functional requirements realm, their partial satisfiability calls for an analysis of a widely debated field of RE, that of non-functional requirements, qualities, and softgoals (see, e.g., [4], [5], [6]). Recent work by Li et al. [7] has provided an ontologically well-founded treatment of this topic:

> "A requirement $r$ that refers to a quality $q$ is non-functional; further, if the quality type of $q$ has an acknowledged shared quality space among the stakeholders, then $r$ is a quality constraint; while if the corresponding quality space is not shared among the RE participants (hence $r$ is vague for agreed success), then $r$ is a softgoal."

This characterization shows how quality constraints and softgoals partition the non-functional requirements domain. Discussing this topic is relevant because the formulation of unbounded requirements in practice may refer to qualities as well; however, unbounded requirements differ from non-functional ones principally, since they explicitly require evidence to increase satisfaction: this trait is not mentioned in this and other definitions of non-functional requirements.

In the KAOS framework [8], goals are "nonoperational objectives to be achieved by the composite system". Unbounded requirements differ because they can be operational (e.g., "Summarize an event's description"), and their satisfaction

depends on the collection of evidence from multiple sources, as opposed to the coordination among agents such as in KAOS.

Pohl [9] defines RE as a process that improves initial, unclear requirements in terms of agreement among stakeholders, formality of representation, and completeness of specification. This is a useful framework for RE, but it is orthogonal to unboundedness: agreed upon, formal, and complete requirements can be still unbounded, such as "Determine if weather will adversely affect an event" (from Table I).

Fuzzy goals [10] are goals whose satisfaction is defined by fuzzy constraints, and have been proposed for self-adaptive systems as a technique for relaxing crisp goals to weaker versions that can be better satisfied by a system (e.g., a goal such as "clean all dirty clothes" would be transformed into "ensure max 5 dirty clothes are not cleaned"). The RELAX framework [11] also exploits similar ideas, and proposes a methodological treatment to do such relaxation. These techniques, despite their usefulness for self-adaptive systems, do not account for requirements unboundedness, and do not support evidence-based satisfaction as our framework does.

## III. Evidential Satisfaction of Unbounded Requirements

The distinction between unbounded requirements and other types of requirements is heavily reliant on the fact that their satisfaction criteria differs. Their *evidential satisfaction* inherently calls for the collection of (partial) evidence through the consultation of multiple providers, that we call *sources*, and that we define as the place of origin of an opinion.

While some sources may reside within the system context (as in traditional software development), the external sources in the Internet offer a clear potential to achieve higher level of satisfaction levels for unbounded requirements. In fact, the reason why the issue of unboundedness has not been a research focus previously may be because, before the emergence and evolution of the Internet, efficient means for collecting evidence from a large number of sources were not available except in specialized contexts (e.g., weather data collection).

In order to determine whether or not an entity is a source, we recommend to employ the "Ask" heuristic [3]: "Can I ask such entity for an opinion concerning the considered task?". Different source types exist that we should distinguish:

- **Algorithm**: it executes *within* the context of the calling system. Consider the Microsoft Word application: invoking (asking) the embedded autosummary function is an algorithm, for a copy of Word is used locally to automatically generate a summary, without involving any sources outside the context of the application.
- **Service**: an algorithm executing *outside* the context of the system under design, such as a third-party web service. For instance, consider an event planner application's function to determine if two events are similar based on their description. In such application, asking for a similarity value to SEMILAR[3] involves invoking a remote service outside the context of the event planner.
- **Person**: a task assigned to one or more persons through a crowdwork platform or through direct communication (e.g., email, instant messaging). For example, the summarization job that we mentioned earlier can be posted as a job on Amazon Mechanical Turk. The idea is that one or more members of the Internet crowd will take up responsibility for that task, instead of relying on a technical implementation to satisfy that requirement.
- **Information**: a corpus of unstructured information that can be queried using a search engine or a crawler and aggregated/analyzed statistically. For example, to check if two event summaries are similar, a solution based on an information source would be to ask Google search if more than 1,000 hits exist of pages that contain keywords from both descriptions.

There is, however, a serious challenge to be dealt with when interacting with sources that are outside the span of control of the system under design – the unreliability of sources. In particular, the accessed services, people or information may become unavailable at unpredictable times, may take longer than expected, may be unable to fulfill a request or may appear to succeed but in fact return partial or incorrect results. To mitigate this challenge, we propose using two reliability attributes of the sources such as measures of availability and credibility to analyze unbounded requirements:

- *availability* is the probability that a source will provide an answer, within a predefined deadline, when invoked;
- *credibility* is a quality metric indicating the average quality of the returned answer, when an answer is returned.

These attributes help evaluate the alternative sources to consult for satisfying an unbounded requirement. For example, consider the summarization of event descriptions. Using the Word autosummarization algorithm has high availability and low credibility: an answer will be returned in most cases, but the result may not be good. On the other hand, asking people through a crowdwork platform has lower availability and higher credibility [12]: when an answer is provided, it will be of relatively high quality due to the intelligent processing that humans are capable of; however, it is possible that noone provides the answer by the given deadline.

Computing availability and credibility is an open research challenge; the literature on trust and reputation can provide useful models [13] to conduct this difficult task.

To better cope with the unreliability of sources (either due to their unavailability or low credibility), one can think of *composing* multiple sources to obtain a higher satisfaction degree of unbounded requirements. While several types of compositions may exist, we identify two main categories here that originate from reliability engineering [14]:

- **Fallback composition**: a number of sources are executed in a sequential order. The $i^{th}$ source is executed only when the $i-1^{th}$ source does not respond within a given deadline, or if its answer does not meet a given quality criterion. The order of invocation of the sources can be either predefined or random.

[3]SEMILAR, www.semanticsimilarity.org, is a semantic similarity engine.

- **Concurrent composition**: a number of sources are called simultaneously, and their results are aggregated to obtain a single answer to the unbounded requirement. This type of composition has many facets to consider:
  - *How long to wait for?* Each of the consulted sources may return no response; as such, a key issue is to avoid situations where the system would wait indefinitely. Different strategies exist: either the computation terminates when a subset of the consulted sources returns an answer, or the termination criterion relies on the quality (credibility) of the combined answer.
  - *How to aggregate the results?* In certain cases, it is relatively straightforward to aggregate the results: for a task that requires a boolean answer, one could simply use majority voting to determine the aggregate answer. In other cases, however, aggregation is tricky: how to combine multiple summaries provided by several sources? No general aggregation algorithm exists, but a possible solution is to rely on another (human) source to perform the aggregation. Notably, the problem involves the resolution of conflicts between the results from multiple sources.
  - *What type of sources?* The choice of the actual sources to combine depends on the task at hand. In case an opinion is required, a viable solution would be to consult many human sources. When different source types differ in terms of availability and credibility (see the example on summarization), it makes sense to combine them in such a way that some result will be obtained (e.g., Word autosummarization), but the quality of the result will be higher in some cases (e.g., when a human provides a summary).

The composition strategies can be employed to decide how to synergistically make use of different sources. For example, the event summary function could be realized through a fallback composition: posting a job on Mechanical Turk with a deadline of two hours, followed by the Word autosummarization if no-one completes the job within the deadline.

## IV. Unbounded Requirements in Real Applications

In this section, we report on our study concerning the existence of unbounded requirements in two real-world software systems: (i) Microsoft Word, an example of a traditional desktop application; and (ii) Spotify, a large-scale (over 30 million songs, over 60 million users) modern, peer-to-peer application for music streaming.

We aim to evaluate the definitional framework of Sec. II for the identification of unbounded requirements (Sec. IV-A), and we discuss the use of strategies of Sec. III as a way to maximize the satisfaction of unbounded requirements (Sec. IV-B).

### A. Identification

We examined Spotify and Word by applying our framework for characterizing unbounded requirements: we employed Def. 1 to determine if a requirement is unbounded, and we evaluated the requirements according to Table I to identify the main reason(s) for unboundedness. For Spotify, we looked at 386 feature requests from the Spotify Ideas platform[4] that enables users to express their wishes for the next releases of the system. For Microsoft Word, we manually searched for functions of the user interface from the 2010 edition that could be identified as unbounded requirements.

*1) Spotify:* Our study shows that unbounded requirements often exist in user feature requests. At least 5% of the 386 scanned feature requests are unbounded (see Table II for a few examples). We also looked at the label that developers assigned to the feature requests. Interestingly, the group of implemented requirements (n = 239) scores the lowest with respect to the number of unbounded requirements; approximately 2% of them can be seen as unbounded. On the other hand, the highest percentage of unbounded requirements was found in the group labeled "not implementable": 28% out of 25 are unbounded. The identified requirements are mostly of three types:

- *Impractical*: although existent, information about upcoming albums and collaborations of singers/players is hard to retrieve. We hypothesize that this issue is common in many modern web-integrated, information-centric applications that provide a vast amount of up-to-date, changing information to the users.
- *Intelligence*: several requirements call for the use of computational/artificial intelligence techniques such as voice recognition (for the scrolling lyrics feature) and recommendation (e.g., for the "discover" feature).
- *Subjective*: some terms lead to different interpretations, such as the notion of "music similarity", which cannot be determined via a unique, objective criterion.

Furthermore, by analyzing developer comments, we found that several requests were often regarded as outside the developers competence, calling for third party software developers. In the stage of implementing, developers often indicated they are unable to fully satisfy the requirement. Other comments described them as infeasible, overwhelming, underspecified, closed for now, or refrained from comments.

*2) Microsoft Word:* Our analysis reveals that unbounded requirements do exist in software in common use. At least 9 functions of Microsoft Word have an unbounded characteristic, with Translate, Grammar check, and Auto-layout among them (see some examples in Table II). The Auto-summarize function, a feature of the 2007 edition that constructed a summary of a document portion on-the-fly, was removed since the 2010 edition. One might surmise that it was removed because the requirement could not be satisfied sufficiently well.

Looking at the requirements in Table II, all of the unbounded feature requirements fall into ungeneralizable, dynamic or requiring intelligence. This may occur because we analyzed currently implemented features for Word, rather than features proposed by end-users. The main take-away from the study of Word is that many functions that are proposed as

[4]https://community.spotify.com/t5/Spotify-Ideas/idb-p/ideaexchange

TABLE II
SOME UNBOUNDED REQUIREMENTS FOR SPOTIFY AND WORD

| Requirement | System | Why unbounded |
|---|---|---|
| Add the date of announced albums | Spotify | Impractical: hard to obtain. Information on release dates is dispersed, and it is practically impossible to obtain all these dates. |
| Show previous collaborations of singers/musicians | Spotify | Impractical: hard to obtain. Many collaborations are not easily accessible (e.g., guest singers). |
| Scrolling lyrics | Spotify | Requires intelligence. This activity requires human judgment to associate lyrics with the voice in the song, especially for slang and dialects. |
| Discover: do not only show me things similar to what I listen to | Spotify | Requires intelligence. This requires humans to find music that one may like, although different from current music |
| Button to filter out explicit songs | Spotify | Ungeneralizable, dynamic. Checking a dictionary would hardly recognize all explicit idioms; moreover, the notion of being explicit changes over time. |
| Suggest similar music | Spotify | Subjective, requires intelligence. To tell if music is similar will require a human judgement. |
| Filter out inappropriate content for the youth | Spotify | Ungeneralizable, dynamic. There are very different rules on inappropriateness and youth depending on the country, and inappropriateness changes over time. |
| Ink to Text | Word | Requires intelligence. Conversion based on handwriting structure (pattern recognition) can produce errors that can be eliminated by using the semantics of the text. |
| Hyphenation | Word | Ungeneralizable. This is because the rules for hyphenation may have exceptions on specific words. |
| Thesaurus | Word | Dynamic. Again since the language is living, new synonyms are always appearing. |
| Translate | Word | Requires intelligence. It is well known that human-level automated language translation has not yet been achieved and good translation requires a knowledge of the text semantics. |
| Compare documents | Word | Requires intelligence. Identifying corresponding parts in documents using only syntax sometimes causes errors. A better comparison is possible when the semantics of text is considered. |
| Auto layout for organization charts | Word | Requires intelligence. For complex charts, producing a good layout requires judgement. |
| Auto-summarize | Word | Requires intelligence. Auto-summarization algorithms can do limited summarization but can be improved if the semantics of the text are used to summarize it. |

standard features of word processors are actually approximations that only marginally satisfy unbounded requirements. For Word, unboundedness seems to be highly related with the fact that the considered functions have to do with language comprehension, where *ambiguity* is a known problem both syntactically [15] and semantically [16].

*3) Remarks:* An obstacle that we experienced during identification is that it is sometimes difficult to tell quickly if a requirement is unbounded; quite often, unboundedness is initially confused with unclarity [9]. However, our framework calls the analysts for a deeper analysis, which helps preventing an early relaxation of the original requirement into a weaker, fully satisfiable version. However, further studies are required to determine how frequent unbounded requirements are, and whether a correlation exists with application type or domain.

### B. Finding Solutions to Unbounded Requirements

We discuss how some of the requirements of Table II can be realized through the composition of multiple sources, in order to increase their satisfaction. For Microsoft Word, we are inspired by its experimental extension Soylent [12] which embeds human computation for certain functions.

*1) Spotify:* Let us consider Spotify's feature request of "filtering out the content that is inappropriate for the youth". At the writing time (June 2015), this requirement is marked as "under consideration" by the Spotify team, and has been in that state since May 2014. This requirement can be refined into two more basic requirements: checking if a song is inappropriate, and detecting if the listener is youth.

Concerning inappropriateness, the content to be examined includes both lyrics and pictures associated with a song. To detect explicit content in lyrics, fallback composition (see Sec. III) can be used. First, one could call an algorithm to compare lyrics against the dictionary to detect any explicit words. If no match is returned, the iTunes search API could be used to determine if the song is explicit. If the song is outside the iTunes catalog, other Spotify users currently playing the song are asked to tell whether that song is explicit. The use of the fallback operator illustrates a gradual invocation of alternative sources in case the previously called ones failed. This operator's strength is the ability to overcome the inherent unreliability of the sources. To check pictures, concurrent composition could be used, by combining the results of Google's Safe Search (a service) and by asking Spotify users (people) to report explicit pictures. A conservative way to aggregate the results is to remove the image if it does not appear in Safe Search's results, or a user marked it as explicit.

Detecting if the listener is youth is also an unbounded requirement (of type impractical). A possible way to maximize its satisfaction is via a fallback invocation: a crawler is asked to retrieve the information from the listener's Facebook profile (Facebook login is allowed in Spotify); if the user has no Facebook or her page contains no information concerning age, then listener herself is asked to confirm she is of legal age.

*2) Microsoft Word:* Take the grammar checking feature, which is currently implemented using only one internal source: an algorithm within the context of Word. Let us consider a variant of this version that uses multiple sources to provide higher-quality grammar checking. We assume that a segment of text has been selected by the user and the grammar-checking function has been invoked.

The text segment is sent concurrently to four sources: the built-in grammar-checking algorithm and three online grammar checking services (for example, grammark.org, gram-

marly.com, whitesmoke.com). Each source will identify fragments of the text that have a grammar problem and provide a corresponding explanation for each of these. One can specify that at least three of these sources must return an answer, so to allow computing the level of agreement of the sources on each identified fragment to determine the confidence in each grammar error: the more sources return the same error, the higher the confidence that is a real error.

In order to further increase the level of satisfaction of the requirement, a fallback composition can be employed that fails-through to a person source if the confidence in the results returned by the automated sources is too low; e.g., if there is too much disagreement among the sources on where the problems lie. An implementation would likely allow the user to decide whether or not to fail-through to the person source since it is more costly and slow (but it would increase satisfaction).

This multiple source approach satisfies the requirement for grammar checking better than the Word grammar checking algorithm alone. Requiring the agreement of multiple grammar-checking algorithms for each grammar problem, including Word's internal functionality, increases the confidence that the identified problems are real. Furthermore, combining explanations from multiple sources will potentially produce better (at worst, redundant) explanations of the identified problems.

Consider the main feature of the Soylent [12] extension of Word: a crowdsourced version of a proofreading application. This is an excellent example of trying to satisfy an unbounded requirement (proofreading) by involving sources of type *people*. Specifically, proofreading is broken into three subtasks assigned to different groups of 3 to 5 people in order to improve the quality of the results: find problem areas, suggest rewrites for these areas and finally, verify that the rewrites fix the problems.

An interesting result of the evaluation conducted by Soylent's authors is that while using the Microsoft Word grammar checker alone caught 30% of the errors and CrowdProof caught 67% of the errors, using both together caught 82%. Thus, a multiple source model that combines the algorithmic grammar checker and CrowdProof improves satisfaction for the considered unbounded requirement.

## V. Conclusion

In this paper, we have introduced *unbounded requirements* as a type of functional requirements that cannot be fully satisfied, either theoretically or practically, and whose satisfaction is increased by gathering evidence from multiple sources of different types: algorithms, services, information, and people.

We postulate that unbounded requirements have to be identified and analyzed adequately, as opposed to labeling them as "unfeasible" or "unclear" and interpreting them differently than the original need (using an approximated version of the requirement that can be fully satisfied).

We have shown the importance of unbounded requirements for Internet-integrated software, and we emphasized how the sources that exist in the Internet—services, people, and information—can be combined to increase the satisfaction. In particular, reliance on the crowd of people in the Internet constitutes a new way of satisfying requirements by relying on humans. For unbounded requirements, this type of source has typically very high credibility (roughly, quality), but low level of availability (due to humans' autonomy).

This paper paves the way for additional work about unbounded requirements. Modeling notations are required to enable representing and refining unbounded requirements. These notations should be complemented with reasoning techniques that enable comparing alternative solutions to the unbounded requirements in terms of their expected satisfaction level (in the spirit of goal analysis [17]). Finally, we need to evaluate empirically to what extent treating unbounded requirements as a separate artifact through industrial case studies.

## References


[1] R. Braden, "Requirements for Internet Hosts – Communication Layers," 1989, Network Working Group RFC 1122. [Online]. Available: http://tools.ietf.org/html/rfc1122

[2] K. Hellenga, "Social Space, the Final Frontier: Adolescents on the Internet," in *The Changing Adolescent Experience: Societal Trends and the Transition to Adulthood*, J. T. Mortimer and R. W. Larson, Eds. Cambridge University Press, 2002, pp. 208–249.

[3] R. Salay, F. Dalpiaz, and M. Chechik, "Integrating Crowd Intelligence into Software," in *Proc. of the ICSE Workshop on Crowdsourcing in Software Engineering (CSI-SE)*, 2015.

[4] I. J. Jureta, S. Faulkner, and P.-Y. Schobbens, "A More Expressive Softgoal Conceptualization for Quality Requirements Analysis," in *Proc. of the Int. Conf. on Conceptual Modeling (ER)*, 2006, pp. 1–14.

[5] M. Glinz, "On Non-Functional Requirements," in *Proc. of the Int. Requirements Engineering Conf. (RE)*. IEEE, 2007, pp. 21–26.

[6] D. Mairiza, D. Zowghi, and N. Nurmuliani, "An Investigation into the Notion of Non-Functional Requirements," in *Proc. of the ACM Symposium on Applied Computing (SAC)*. ACM, 2010, pp. 311–317.

[7] F.-L. Li, J. Horkoff, J. Mylopoulos, R. S. Guizzardi, G. Guizzardi, A. Borgida, and L. Liu, "Non-functional Requirements as Qualities, with a Spice of Ontology," in *Proc. of the IEEE Int. Requirements Engineering Conf. (RE)*. IEEE, 2014, pp. 293–302.

[8] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-Directed Requirements Acquisition," *Science of Computer Programming*, vol. 20, no. 1, pp. 3–50, 1993.

[9] K. Pohl, "The Three Dimensions of Requirements Engineering: A Framework and Its Applications," *Information Systems*, vol. 19, no. 3, pp. 243–258, 1994.

[10] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy Goals for Requirements-driven Adaptation," in *Proc. of the IEEE Int. Requirements Engineering Conf. (RE)*. IEEE, 2010, pp. 125–134.

[11] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "RELAX: Incorporating Uncertainty into the Specification of Self-adaptive Systems," in *Proc. of the IEEE Int. Requirements Engineering Conf. (RE)*. IEEE, 2009, pp. 79–88.

[12] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: A Word Processor with a Crowd Inside," in *Proc. of the ACM User Interface Software and Technology Symposium*. ACM, 2010, pp. 313–322.

[13] I. Pinyol and J. Sabater-Mir, "Computational trust and reputation models for open multi-agent systems: a review," *Artificial Intelligence Review*, vol. 40, no. 1, pp. 1–25, 2013.

[14] K. C. Kapur and M. Pecht, *Reliability engineering*. Wiley, 2014.

[15] M. C. MacDonald, N. J. Pearlmutter, and M. S. Seidenberg, "The Lexical Nature of Syntactic Ambiguity Resolution," *Psychological review*, vol. 101, no. 4, p. 676, 1994.

[16] J. Rodd, G. Gaskell, and W. Marslen-Wilson, "Making Sense of Semantic Ambiguity: Semantic Competition in Lexical Access," *Journal of Memory and Language*, vol. 46, no. 2, pp. 245–266, 2002.

[17] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with Goal Models," in *Proc. of the Int. Conf. on Conceptual Modeling (ER)*, 2003, pp. 167–181.