2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement

A Complexity Measure for Textual Requirements

Vard Antinyan[1], Miroslaw Staron[1], Anna Sandberg[2], and Jörgen Hansson[3]

[1] Computer Science and Engineering, University of Gothenburg | Chalmers
[2] Ericsson, Sweden [3] School of Informatics, University of Skövde
SE 412 96 Gothenburg

Abstract**—**Unequivocally understandable requirements are vital for software design process. However, in practice it is hard to achieve the desired level of understandability, because in large software products a substantial amount of requirements tend to have ambiguous or complex descriptions. Over time such requirements decelerate the development speed and increase the risk of late design modifications, therefore finding and improving them is an urgent task for software designers. Manual reviewing is one way of addressing the problem, but it is effort-intensive and critically slow for large products. Another way is using measurement, in which case one needs to design effective measures. In recent years there have been great endeavors in creating and validating measures for requirements understandability: most of the measures focused on ambiguous patterns. While ambiguity is one property that has major effect on understandability, there is also another important property, complexity, which also has major effect on understandability, but is relatively less investigated. In this paper we define a complexity measure for textual requirements through an action research project in a large software development organization. We also present its evaluation results in three large companies. The evaluation shows that there is a significant correlation between the measurement values and the manual assessment values of practitioners. We recommend this measure to be used with earlier created ambiguity measures as means for automated identification of complex specifications.

Keywords**—**measure, requirement, quality, complexity, automation

I. Introduction

Accurately specified requirements help software designers to understand what exactly the functionality is to be developed. They also help software testers to understand the exact outcome of tests, when verifying the developed functionality. Oppositely, if the requirements are inaccurate, software designers and testers will need further clarification, which slows down the design and verification process. Additionally, inaccurate requirements increase the risk of late design modifications [1], [2], [3], which triggers project delays. A traditional strategy for reducing this risk is conducting requirements reviews. The aim of reviews is to detect inaccurate specifications and improve them before they are used for design and verification. There are several papers indicating that early and fast improvements significantly aid the design and verification process [4], [5], [6]. There are frameworks for improvements also [7], [8], [9], however, in *large continuous software development projects* manual reviews before the start of software design often is not possible because of two main reasons: 1) the amount of requirements are too large, typically several thousand, the reviews of which is a tiresome and effort-intensive task [10], [11], [12], [13], [14]; 2) all development activities are continuous [15], so it is not an optimal solution to stop the development process and wait until the reviews are finished. In this situation practitioners find tool support is a key for efficient reviewing process, as the size of software products grow [5].

To enable tool support in requirements reviews researchers have introduced several measures for text based requirements, e. g. [16], [17], [18]. Majority of these measures are focused on detecting ambiguous expressions in requirements' texts, such as weak verbs, qualitative adjectives, unjustified use of pronouns, etc. While *ambiguity* is one important attribute that affects the understandability of requirements, it is not a decisive one. There is also another attribute that significantly affects requirements' understandability, and that is the *complexity* of requirements. Ideally a requirement should be one atomic statement the specification of which indicates one actor performing one function. However, as practice shows, this is often not the case. In many practical examples, it is not easy to effectively isolate "one actor, one function" concept into distinct requirements and therefore a requirement might contain several actors performing several tightly related functions. Because of this, the complexity of requirements increases, and thus the understandability decreases.

In order to empower requirements complexity measurement we conducted this research. Our aim is to introduce a measure that quantifies the complexity of requirements, which can support automated review process. The research question we address in this paper is:

How can we assess complexity of textual requirements?

In this paper we define and present *conjunctive complexity* of textual requirements. The *conjunctive complexity* measure first was designed through conducting an action research project in one software development organization, and then it was evaluated in three other software development organizations. The measure is based on morphological conjunctions of natural language text and indicates the amount of actions (actors) and their relations in a requirement. The evaluation results show that the complexity measurement values and manual assessment values of requirements understandability by human assessors have a significant level of agreement. Additionally we found that, the measure is robust in assessing different types of requirements across the organizations. These results support the generalizability of the measure, which is particularly difficult to achieve for text-based measurement entities.

The rest of the paper is organized as follows: Section 2 presents the collaborating companies. Section 3 defined the concept of requirements complexity. Section 4 describes the action research approach that we employed for designing the measure. Section 5 presents the measure, brings relevant practical examples, and presents the measurement method. Section 6 describes the method of evaluation in three companies. Section 7 presents the evaluation results. Sections 8 and 9 discuss the validity threats and the related research respectively.

II. Collaborating Companies

Three companies were involved in this research: Volvo group, Saab, and Volvo Car Group. First we employed an action research project in an organization in one of the companies to design the measure. Then we evaluated the measure in three other organizations in three companies respectively. Unfortunately we cannot reveal the name of the organization that helped as in conducting the action research project. Similarly we cannot provide evaluation results mapped on company names. Companies considered this information confidential. The three collaborating companies were as follows:

Volvo Group is a Swedish company that develops and manufactures trucks, buses, and construction machines. The company develops ECUs (electronic control units) for these vehicles and machines. An ECU is a computer which controls a specified functionality for a vehicle automatically. Such functionalities can be, for example, *climate control* or *break control*. The organization that we collaborated with in Volvo Group develops software for the main ECU of a vehicle.

Saab Defence and Security is a Swedish company which develops and manufactures products and services for military defence and civil security. One of the security equipment is a ground radar system. The collaborating organization develops software system for the radar system.

Volvo Car Group is a Swedish company that develops and manufactures vehicles in the premium segment. Similar to Volvo Group the company develops ECUs, which perform variety of functionalities for cars. Our collaboration unit develops ECU for climate control of the car.

We analysed requirements of two abstraction levels, subsystem level and component level. Requirements of component level are more detailed and have a bigger size of text. They are the most detailed level of requirements and are directly used by software designers for coding. Requirements of subsystem level are more general in their description and are smaller in size.

In two of the collaborating companies the requirements that we used were of component level. In the third company we used requirements of subsystem level. The action research project was conducted in an organization that had requirements of component level. Requirements in all the companies were written in English.

III. Defining Requirements Complexity

The term of complexity has been defined and widely used in many disciplines, to describe an intrinsic quality of a system, which strongly influence human understandability of how the system operates. Thus far there has not been any generally accepted definition of complexity, which supports its measurement in a standard way. So every discipline has its own approximation of defining what complexity is and how to quantify it. The standard books of software measurement do not really explain where complexity emerges from. In IEEE standard computer dictionary complexity is defined as “the degree to which a system or component has a design or implementation that is difficult to understand and verify” [19]. This definition is of little help for understanding how to measure complexity, because it is based on the cognitive capability of the readers. Similar definitions are provided by Zuse [20] who defines complexity as the difficulty to maintain, change, and understand software, by Fenton and Bieman [21] who view complexity as the resources spent on developing (or maintaining) a solution for a given task, and by Basili [22] who defines complexity as a measure of the resources allocated by a system or human while interacting with a piece of software to perform a given task. All of these definitions specify what complexity causes, such as difficulty in understanding or resource consumption, however they do not elaborate where complexity comes from. This fact is noticed by Briand, et al. [23] who argues that complexity should be defined as an intrinsic attribute of a system but not its perceived difficulty by an external observer. In this case it would be easier to understand where complexity comes from and how to quantify it. Probably the most suitable definition of complexity, which eases its quantification is provided by Rechtin and Maier [24] according to whom:

A complex system has a set of different elements so connected or related as to perform a unique function not performable by the elements alone.

This definition of complexity implies that complexity emerges from and is magnified by the quantity of elements and their interconnections in a system. If this quantity increases the understandability and modifiability of the system decrease, because one needs to comprehend how all the elements are connected together in order to perform a unique function. We found this definition of complexity is suitable for defining requirements complexity, because it explains where complexity emerges from, thus enabling its quantification. We adapt the original definition to textual requirements the following way:

The complexity of a textual requirement is defined by its number of different actions so connected or related as to indicate a unique function.

As the definition shows, such complexity is not concerned with the syntactic complexity of the text. This means we define complexity from the standpoint of system design, hence, this definition is not anyhow concerned with the syntactic structure of the requirements text. However, we should notice that if there are more actors (actions) and relations described in a requirement's text the syntax of that text will become more complex. To this point we provided a definition of textual requirements' complexity, which assumes that complexity of a textual requirement can be quantified by quantifying the number of related actions in a requirement. In the next section we present the process of how we examined the possibilities of such quantification in a large software development organization.

IV. Designing the Measure

We employed action research methodology for designing the measure [25], [26]. This means instead of designing the measure in isolation and then evaluating its validity in a company, we decided to design the measure in the company with help of systematic feedbacks of software engineers, who work closely with software requirements. We considered that software engineers' feedback on the designed measure would allow us to qualify its intuitiveness and understand its effectiveness in practice. Based on such feedbacks we could refine or redefine the measure if necessary, in order to both improve the measure and evaluate the measurement feasibility in practical context.

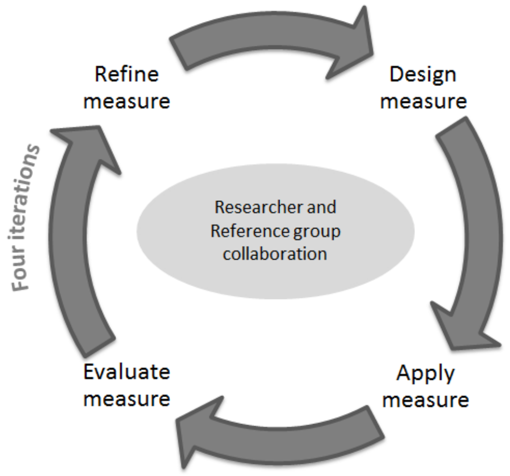

Figure 1 Action research cycle for designing the complexity measure

To conduct the action research project we asked for a group of engineers from the collaborating organization for helping us with their feedback. We refer to this group of engineers as reference group. The reference group consisted of a design architect, a software designer (developer), and a test leader. All three of them had core knowledge of the developed product and were directly using software requirements for software design and verification. We had both regular monthly meetings and many informal meetings with the reference group throughout the research project. Figure 1 presents an overview of the action research cycle that we employed. The cycle starts with the *design* of the measure. When the initial version of the measure is designed we *applied* it on the entire set of requirement in the organization. Then we, with help of reference group, qualitatively *evaluated* the measure considering several practical examples from the measured requirements. Finally, the measure or measurement method was *refined* based on the evaluation result, in order to perform more accurate measurement. The cycle was iterated four times, which means that every time that we were refining the measure, there was a need for redesigning it and then the cycle was iterated again. In the end of the fourth iteration we stopped the process, considering that we have reached the limit of refining the created measure. The detailed process of each activity in the cycle is presented in the next four subsections.

A. *Design Measure*

To design a complexity measure, one needs to understand what characteristics of textual requirement make the requirement complex. In order to identify such characteristics we decided to start with exploring several requirements that the reference group was working with. Therefore we asked the engineers of the reference group to provide us with 20 requirements in such a way that 10 of them they had found were difficult to understand and the other 10 they had found were easy to understand. We explored morphological and syntactical features of these requirements to find out what features make one requirement easy to understand and another requirement difficult to understand. We also asked reference group engineers to explain the reasons behind the understandability of these requirements. Based on such analysis we found the linguistic characteristics that most likely have indication of complexity. Grouping and counting such characteristics determined the initial version of the *measure* and *measurement method* (see ISO/IEC 15939).

B. *Apply Measure*

When we had the initial version of the measure and the measurement method, we designed a tool for parsing the entire set of requirements and conduct measurements. In the organization the requirements were stored in a requirements management system. The system had the possibility of extracting all requirements in a structured file. Usually RIF (Requirements Interchange Format) files were used for communicating requirements between different organizations or parties, so we used this format. The file contains much information about the requirements, which is, requirements name, body text, version, number of revisions, variants, etc. We extracted all requirements of our collaboration unit into such a file and conducted measurements. The measurement results were stored in a comma separated (csv) file, where the fist column presented the names of requirements and the second column contained the measurement values. Higher measurement values show higher complexity according to the measure.

C. Evaluate Measure

After obtaining the measurement results we examined several requirements with both high and low measurement values to see whether they have similar characteristics as the first 20 requirements provided by the reference group. At this point, we, researchers selected another set of 20 requirements, 10 of them having high measurement values and the other 10 having low measurement values. We deliberately selected these requirements in such a way, that their descriptions were perceived to be morphologically or syntactically different from the first set of 20 requirements that were provided by the reference group. Then we presented and discussed these requirements with the reference group engineers. The engineers explored the presented requirements one by one and expressed their insights on whether the measurement values of requirements are reasonably congruent with their perception of complexity. This tactic permitted us to find out whether the designed measure was specific for measuring the complexity of the initially provided 20 requirements or it is capable of capturing complexity of other requirements as well.

Besides investigating the generality of the measure we also discussed how accurately the measurement method can be applied for calculating complexity of requirements. Practically this means that we checked how accurately we could calculate the number of interactions between actors by applying the designed measurement method.

D. Refine Measure

Based on the discussion results obtained from evaluating the measure we could assess the possibilities of improving the measurement method. For example, we might decide that certain characteristics of requirements should be considered and parsed for improving measurement. Oppositely we might also decide not to consider some other characteristics because they introduce inaccuracies in the measurement. If there was a room for improvement of the measurement method we iterated the action research cycle once again. This means that after refining the measure we conducted measurement again, selected another set of 20 requirements with the same rules as described in the previous section, and conducted qualitative validation with the reference group. Overall we needed four iterations in this study, until we were satisfied with the qualitative evaluation results and decided to evaluate the measure quantitatively. It is important to mention, that in every iteration we kept all previously selected sets of requirements in the evaluation loop, so we could make sure that the refined version of the measure does not affect the previously evaluated results.

V. Results of Designing the Measure

Let us remind the reader that the complexity measure that we intended to define aimed at assessing the number of actions or relations in a textual requirement. There were several possibilities of assessing this number in our vision and we tried all of them to understand which one is the most accurate for such an assessment.

A. Modal Verbs

The first possibility that we considered for complexity measurement is to count the number of imperative modal verbs. Such verbs are “shall”, “should”, “must”, “ought to”, “will”, “have to”, etc. First, it is intuitive to try this measurement because indicates an action that should be implemented in a requirement. Second, this measure has been considered in the literature as well [16], so it was interesting to see whether such a measure could be of effective use. And third, it is easy to accurately count modal verbs in a text. Thus the first possibility of a complexity measure was based on the modal verbs, and the measurement method of such complexity was to count the number of modal verbs in a requirement. In the first iteration of the action research cycle we conducted measurement based on this consideration and collected the results. The results showed that the count of modal verbs is not accurately representing the number of actions described in a requirement. The discussions with the reference group showed that the count of modal verbs often does not indicate the perceived complexity. In many cases there were several actions described in a requirement without using modal verbs, so generally the count of modal verbs underestimated the number of actions specified in a requirement. A clarifying example is presented in **example 1.**

Example 1

> When ContainerType changes to “not valid” then ContainerCapacity should be set to the last value as long as VolumeReset is requested.

The example shows that there are three distinct actions (and actors) in the requirement: 1) ContainerType changes to “not valid”, 2) ContainerCapacity is set to the last value, and 3) VolumeReset is requested. However the text contains only one modal verb – “should”. Consider the **example 2**.

Example 2

> When RunningMode makes a transition from Running to Living or Accessory while Control Mode is other than PK then light control mode shall be forced to Zero.

There are three related actions in this requirement: 1) RunningMode makes a transition, 2) Control Mode is other than PK, 3) light control mode shall be forced to Zero. We can also observe that there are more than three actors in the requirement, however the requirement contains only one modal verb (“shall”). We explored many examples and found that the count of modal verbs is not an accurate measurement of complexity. This result is intuitive to some extent, because the language itself naturally allows avoiding modal verbs by using other linguistic constructs.

B. Separation Punctuations

In the first iteration we also considered counting the number of separation punctuations, such as commas “,”, colons “:”, and semicolons “;”. This measurement was done in parallel with the measurement of modal verbs. The motivation was that the number of such punctuations can approximate the number of actions in a requirement. However, after conducting the measurements, we realized that such an approximation was inaccurate. The problem is that the software engineers are not always paying strong attention on whether a requirement’s text is grammatically correct, so some of the needed punctuations were often missed. Besides, there was another important factor also, which obsoleted the use of punctuations (and modal verbs): many requirements were not written as sentences, and could not be grammatically assessed, therefore they often did not use punctuations. To clarify this statement, consider the following example:

Example 3

If EPBFeedRequest is received to a value different than "Not Available" **(1)**
    - EPBFeedRequest is equal to "P-feed Not Requested" **(2)**
  or
    - Active Mode goes to Hibernate **(3)**
If EPBFeedRequest was never received **(4)** or received as "Not Available" **(5)**
    - ActuationStatus is equal to "Stable" **(6)** and EPBBackupMode = 0 **(7)**
  or
    - Active Mode goes to Hibernate **(8)**

This requirement is not grammatically assessable, because it is written like a pseudocode. As we have numbered in the example, there are eight actions in the requirement, however, there are neither modal verbs nor separation punctuations. We observed that there are many similar examples among the requirements and that the modal verbs and punctuations are inaccurate measures of complexity. More importantly, such requirements that had many modal verbs were not necessarily perceived as complex by the reference group engineers. A typical example of such a case was a large requirement which contained bulleted points. Every point was a simple specification of functionality that could be developed alone, however, as software engineers explained, they put tightly related specifications into one requirement as bulleted points, because it increases the understandability of the context. Oppositely, there were many requirements for which the measurement values were low, however they were perceived complex and difficult to understand (example 3).

In the end of the first iteration we had discussed over 40 requirements with the reference group and many other requirements by ourselves. At this point we realized that there is another linguistic construct which might be a good measure of complexity: this was the number of conjunctions in a requirement.

C. Conjunctions

Conjunctions are grammatical constructs that connect words, clauses, or sub-sentences. Examples of conjunctions are “and”, “or”, “if”, “whenever”. In the second iteration of the action research cycle we measured the most known conjunctions in requirements. We investigated the previously examined set of requirements again and realized the number of conjunctions is a good approximation of the number of actions described in a requirement. If we observe the previously presented examples of requirements (example 1 through example 3) we can see that the number of conjunctions is closely approximating the number of actions in the requirements. In **Example 1** there are three conjunctions – “when”, “then”, “as long as”. As we noticed earlier the same example has descriptions of three distinct actions. **Example 2** also contains three distinct actions. The same example contains four conjunctions – “when”, “or”, “while”, “then”. The additional fourth conjunction (“or”) in this requirement connects two states, “living” and “accessory”, which is also a manifestation of complexity. The requirement of **Example 3** contains eight actions. Meantime it also contains six conjunctions. As we can notice the number of conjunctions is not exactly equal to the number of actions that we identified, however, they are closely approximating each other. Investigating more examples we found that the number of conjunctions is a good approximation of the number of relations described in a requirement.

In the third iteration we used online English grammar books to find the exhaustive list of English conjunctions and conduct measurements again. After conducting the measurements, we re-evaluated the number of conjunctions as a complexity measure. The evaluation showed that there are several conjunctions, which are not always used as a conjunction, and therefore introduce inaccuracies in the measurement. Examples of such conjunctions are: “than”, “that”, “because” and “so”. We excluded these conjunctions from measurement method. We also did not use compound conjunctions such as “even if”, “even though”, etc. because they contain simple conjunctions “if” and “though”, which were already included in the measurement.

After the aforementioned refinement of measurement method we determined the new list of conjunctions which should be considered in the measurement. The list contained 27 conjunctions. The fourth iteration showed that this list of conjunctions is the most accurate list for measurement. Each of the 27 conjunction was examined in many examples and in all cases they indicated connection of actions or actors. Moreover, based on the feedback of reference group, we found that the high values of the number of conjunctions were usually associated with the perceived complexity of requirements. At this point we decided that the measure is good enough for being formalized and evaluated quantitatively. In the next subsection we define a complexity measure based on the number of conjunctions.

D. A Complexity Measure for Textual Requirement

Concluding the results that we presented in the previous three subsections we define a complexity measure for textual requirements:

Definition: Conjunctive complexity (NC) of a textual requirement is the complexity that emerges due to the number of conjunct actions described in the requirement's text.

The results presented in the previous section showed that *conjunctive complexity* of a requirement can be calculated by counting the number of specified 27 conjunctions in the requirement's text. Table 1 presents the denotation and measurement method for conjunctive complexity. As the table shows, in order to calculate the conjunctive complexity of a textual requirement, we need to count the overall occurrences of any of the 27 conjunctions in the text. The conjunctions presented in the table cannot be viewed as an absolute list for complexity measurement, however it is an exhaustive list for conducting measurement in practice.

Table 1 Denotation and measurement method for the complexity measure

Name	Denotation	Measurement method			
Conjunctive complexity	NC – number of conjunctions	Count the overall number of occurrences of the following 27 conjunctions			
		And After Although As long as Before But Else	If In order In case Nor Or Otherwise Once	Since Then Though Till Unless Until When	Whenever Where Whereas Wherever While Yet

Two examples of NC calculation are presented in **examples 4** and **5**.

Example 4

If a request for RL Mode is received **while** RL Mode is already requested, **then** behavior of signal RL_Mode_rqst shall still be as default.

NC = 3

Example 5

When BOC_Mode_rqst is set to BO mode **or** CO mode all exterior lightning must be shut off immediately

NC = 2

Fewer conjunctions in a requirement show simplicity of that requirement. As the reference group engineers mentioned, ideally software designers and testers prefer no conjunction or maximum one conjunction in a requirement's text. However, the practice shows that the number of conjunctions in a requirement sometimes can exceed ten, which increases the complexity and jeopardizes the understandability of the specification.

We must notice that an amount of conjunctions in a single sentence has a greater effect on the understandability of a requirement than the same amount of conjunctions distributed in several sentences. This is intuitive, because one may think of a sentence as an atomic specification. However, we also observed that, even if the of a requirement contains several sentences, the overall large number of conjunctions still indicates decreased understandability of the specification. This is probably due to the fact that several sentences in a single requirement have strong conceptual connections anyway, so the atomicity of sentences is compromised and thus the overall large number of conjunctions still indicates decreased understandability of a specification. Whether conjunctions distributed in several sentences shall be calculated separately or together is still an inconclusive issue for us. Because the overall number of conjunctions still indicates the complexity of requirements we decided not to separate a requirement into several sentences as several measurement entities. However, we still think that this issue is worth to be investigated and more conclusive answers needs to be provided based on evidence.

Finally we should mention that counting the number of conjunctions is quite simple and straightforward and therefore we can measure NC accurately.

VI. Evaluating the Measure

Once the measure was defined we decided to evaluate it quantitatively. Particularly we were interested in how well the measure can predict such requirements that are perceived as "difficult to understand" by software engineers. In order to conduct such an evaluation we needed manual assessment results of software engineers. Then we could correlate measurement values with the manual assessment results and the correlation coefficients would show the level of agreement between the measurement values and the manual assessment values.

At this point the action research project was finalized, and we finished our collaboration with the reference group in the scope of this study. To obtain manual assessment data we contacted to three software development organizations in Volvo Group, Volvo Car Group, and Saab (1 organization per company). In every organization we had a collaborating person who helped us in acquiring the necessary data for evaluation. In organization 1 our contact person was a senior software designer. He invited a software designer and a tester for helping with manual assessments. In organization 2 we collaborated with a requirements manager. He invited two software designers for manual assessment. In organization 3 we collaborated with a design and verification process coordinator. He invited a requirements analyst and a technical assistant in requirements specification. All six engineers that were invited for manual assessment were deliberately chosen in a way that they had many years of experience (> 8 years) and fundamental knowledge in requirements management and use. In every company the evaluation was conducted separately, thus in every company a pair of engineers participated in the evaluation process.

A. *Selecting Requirements for Manual Assessment*

After inviting engineers for manual assessment we needed to select a sample set of requirements in the software product of a targeted organization. The following procedure was employed for selecting sample data:

- Use the measurement tool to conduct measurements and calculate NC values for the entire set of requirements of the product
- Rank all requirements based on the NC values in descending order
- Divide the entire ranked list into 30 equal groups of requirements in such a way that the first group of requirements have the highest NC values, the second group has next highest values, etc. The last group of requirements has the smallest NC values
- Randomly select one requirement from each group. This results in getting a set of 30 requirements

The ranking and grouping are done deliberately to ensure that the selected set of requirements has variety of NC values. If we did not do this step and just randomly select 30 requirements we would end up in having most of the 30 requirements with low NC values. This is because in the entire set of requirements, which are several thousands, 200-300 were estimated to be complex, thus the random selection would cause the most of the 30 requirements to have low NC values. This, in its turn, might artificially increase the correlation between the manual assessment values and NC values because most of the 30 requirements were likely to have low NC values and low manual assessment ranks. The number 30 for the sample set was chosen considering two factors:

- There should be enough requirements for representing the requirements population
- There should not be too many requirements for the manual assessors, so they get tired and provide inaccurate assessment results

After the selection of the sample set we asked the invited software engineers to rank the understandability of these requirements in a Likert scale of [1, 5]. We also provided with explanations of the ranks, so they could better understand how the rankings should be performed. The explanations of the ranks are provided in Table 2.

Table 2 Assessment ranks and their explanation

Rank	Description
1	It is absolutely easy to understand this requirement
2	It is rather easy to understand this requirement
3	This requirement can be improved to make it easy to understand
4	This requirement should be improved, because it is hard to understand
5	This requirement must be improved, it is not possible to understand

The engineers who assessed the sample set did not know the measurement values of requirements and did not have any kind of participation in the measurement design. The only thing they knew was that they were asked to assess 30 requirements with the given ranking scheme to help us in a study.

For each of the company we collected assessment results separately, thus resulting in three sets of requirements with manual assessment results. In each set of requirements we had a pair of assessment results. In the next section we explain how these results were used for the correlation analyses.

B. *Evaluating the Measure against the Manual Assessment Results*

In order to get a single manual assessment value per requirement we averaged the assessment results of two engineers. If we denote the given ranks of the first engineer by QI_{E1} and the ranks of the second engineer QI_{E2}, then the average rank (QI_E) was calculated as $QI_E = (QI_{E1} + QI_{E2})/2$.

To evaluate the agreement between NC measurement values and QI_E manual assessment values we conducted correlation analyses between them. The population of data was not normally distributed, thus making Spearman rank correlation more appropriate to use, however we also provide Pearson correlation coefficients as an alternative statistic of comparison.

We also assessed the agreement between the two engineers' assessments, so we could understand whether the manual assessment results are reliable enough against which a measure is evaluated. If the engineers poorly agree together then there is a risk that the requirements understandability is not a well understood concept, or different people understand it differently, so more evaluation is needed. The agreement between the manual assessors was evaluated by the Kendall's tau correlation coefficient. The evaluation results are presented in the next section.

VII. Evaluation Results

The correlation analyses results between NC and QI_E for three companies are presented in Table 3. The first numbers in the cells are Pearson coefficients, the second numbers are Spearman coefficients. Because the companies did not want to have results mapped on their names we conventionally used X, Y, and Z names for the companies. Generally the correlation coefficients show significant association (>0,5) between NC and QI_E values. Spearman coefficients for companies X and Z are strong (0,77 and 0,65). Correlation looks the strongest for company X. The comparably weaker correlation for company Y can be explained with the fact that the requirements of company Y were of subsystem level. Such requirements were usually smaller in size and contained less technical descriptions. Oppositely, the requirements of company X and Z were of component level. They were usually bigger in size and contained detailed descriptions of implementation. Such requirements could contain pseudocode, tables, names of signal, names of states, camel-cased words. etc.

Table 3 Correlation results between the manual assessment and measurement results

	QI_E (X)	QI_E (Y)	QI_E (Z)
NC	**0,74 / 0,77**	0,54 / 0,54	0,58 / **0,65**
p value	< 0,001/0,001	< 0,000/ 0,001	0,001/0,005

These results show that even though the NC measure is not a very strong indicator it nevertheless has significant correlation with the perceived understandability of requirements and can be used for detecting complex requirements. The measure understandably does not purport to be an ultimate measure for assessing understandability of requirements. Moreover, we urge it to be used with the existing ambiguity measures in the literature, because we believe that a combination of two-three good measures can provide strong indication on requirements internal quality.

The results also indicate that NC measure is likely to be a better measure for a lower level of requirements, however we consider that more evaluation is needed to have a clear understanding of this distinction.

Table 4 presents the results of agreement of engineers' assessments for all three companies.

Table 4 Agreement between the engineers' assessments

Company	X	Y	Z
Kendall's tau	0,56	0,61	0,75
P value	0,001	0,000	0,000

The table shows that there is a significant agreement between the manual assessors which means that the manual assessors significantly agree on the criteria of requirements understandability. This increases the chance that the evaluation results obtained and presented in Table 3 are true.

VIII. Threats to Validity

Action research is inherently prone to construct validity threats if it is used for designing a measure. This is because a measure is a means of quantification of a phenomenon, while designing a measure by relying on action research as a qualitative methodology can be a source of validity threat. So why did we choose action research in this study? The practicality of any measure is of the highest importance after it has been designed, and such practicality can be effectively assessed in the context of its application [27]. We chose action research because if a measure is designed in the context of its intended application than there is a bigger chance that the measure is successful. However this does not mean that the validity threats of designing the measure is overcome. In fact we needed to realize several important steps for insuring the veracity of the designed measure. Most importantly we involved reference group engineers in the design process. It is important to understand whether any candidate measure proposed by us is perceived as a good indicator of complexity. We tried to include as many requirements as possible (about 100 of them) in the analysis phase, so the measurement values could be closely observed on these requirements and understood what actually they indicate. There were many other requirements that we researchers observed closely without the reference group, to ensure that the proposed measurement method is rigorous. The list of 27 conjunctions that we have proposed for calculating NC measure is by no means the perfect or most optimal list for NC measurement but they are iterated and tested deliberately on over 100 requirements of a real product, which makes it reasonably trustworthy list. More evaluation surely would bring more value, but it will remain for further studies.

A particular construct validity issue is the consideration of NC measurement on a requirement that has several sentences. What the results showed so far is that the NC measure is a stronger indicator for one sentence than for a whole requirement composed by several sentences. However, the results also showed that several sentences in a requirement are often logically connected anyway, so the overall NC value for the whole requirement still is a sensible number, indicating requirement's overall complexity. This issue was difficult to address because in order to measure the complexity of separate sentences one needs to separate sentences accurately in the first place, and such accuracy was never achieved in our study. Even if we consider that the necessary accuracy could be achieved we need to consider how overall requirements complexity should be calculated based on the complexity numbers of several sentences. This issue becomes even more difficult to address for requirements of component level, where there are no clear definition what a sentence is. There were many requirements where it is not possible to define what is a sentence because of many tables, bullet points, pseudocode-like writing etc. This issue was not possible to exhaustively examine in this study, so we decided to postpone it for our follow-up studies.

A point of concern with the design of this measure is the measurement scale. The number of conjunctions itself is a ratio scale, meaning that it has a point of absolute zero, and can be meaningfully divided and multiplied. However, the purpose of this number is to represent a perceived complexity of a requirement. Hence, it is important to understand whether the perceived complexity is proportionally increasing or decreasing with the changing number of conjunctions. For example, if every time the number of conjunctions is multiplied by two, can we say then that each time the perceived complexity is doubled? This question unfortunately is not investigated in this paper, albeit doing so would add deeper understanding of the measure.

Repeatability of research is one of the most important factors for external validity. Repeatability is particularly problematic for action research projects, due to the complexity of studied systems and human interference. Therefore we consider *recoverability* of our research to be the guarantor for external validity [28]. In this study we have developed a complexity measure in collaboration with one plus three software development organizations: in the first organization we conducted an action research project for designing the measure. Here, the measure was evaluated qualitatively by a reference group. The measure was also quantitatively evaluated in the three organizations. We believe that the amount of evaluation that we have performed is enough to ensure that the obtained results are likely to be generalizable, and it is worth for evaluating this measure in other contexts as well. Therefore we encourage other researchers to apply NC measure in other contexts, with combination of other measures, and report results on that.

IX. Related Work

One of the early studies that measures quality of textual requirements is conducted by Davis, et al. [29]. The study proposes 24 quality properties and measures 18 of them. The study was one of the first studies in the area of requirements measurement, so it is rather a guide for defining measures for and conducting measurement on for textual requirements. In another early paper reported by Hyatt and Rosenberg [17] we can find measures for identifying ambiguous phrases of textual requirements. Authors also used these measures in a software metrics program.

There are also recent papers addressing the issue of requirements' measurement. For example Vlas and Robinson [30] reports measures for requirements classification. They first define textual patterns and then based on these patterns automatically classify requirements into predefined requirements groups. Huertas and Juárez-Ramírez [31] present an automated tool for measuring ambiguity and atomicity of textual requirements. Kasser, et al. [32] define measures for assessing the quality of requirements. The identify words and phrases that introduce uncertainty to the requirements text.

Gleich, et al. [18] examined the ambiguous phrases in textual requirements and developed a tool for ambiguity detection. The study is notable by the fact that the ambiguity patterns are explained with a great detail, and the results are evaluated with both software engineers and academicians. Fabbrini, et al. [33] present a tool (QuARS) for automatic identification of low-quality requirements. The tool is based on a number of measurements subsumed into four quality categories: testability, completeness, understandability, consistency. Génova, et al. [16] propose a framework and tool support (Requirements Quality Analyser) for assessing textual requirements' quality. They measure such properties as *size*, *readability*, *punctuation*, *imprecise terms*, *verbal tense*, *number of versions*, *degree of nesting*, *overlapping*, and *dependencies* of requirements. The aforementioned two studies does not specify how exactly the measures are calculated and evaluated. They qualitatively evaluate the tools based on the feedback of companies that use it. As our results showed, *punctuations* are poor indicators and introduce inaccuracies in the measurement. Parra, et al. [34] use measures presented in [16] to evaluate the *correctness* of requirements. They train a classifier based on the measures and experts' manual classification to identify "correct" and "incorrect" requirements.

X. Conclusions

In this paper we presented a complexity measure for textual requirements. The measure aims at supporting automatic identification of such requirements that are difficult to understand because of complex specifications. As in large software development organizations the number of textual requirements can be several thousand, automatically detecting such specifications can be of great support for software engineers. It helps them to conduct early improvements and reduce risks of late design modifications.

The measure, conjunctive complexity, was designed and qualitatively evaluated through an action research project in a large software development organization. Later it was also quantitatively evaluated in three other organizations.

The results show that the measure is very simple and easy to calculate in practice. The evaluation results showed that there is a significant agreement between measurement values and manual assessment values of software engineers, indicating that the measure can be used to detect such requirements that are perceived to be difficult to understand. We recommend using this measure with the previously proposed ambiguity measures to strengthen the accuracy of understandability assessment.

Acknowledgment

We express our thanks to the companies and universities for their support in the study. This research has been carried out in the Software Center, Chalmers, University of Gothenburg, Ericsson, Volvo Group Truck Technology, Volvo Car Corporation, and Saab Defense and Security.

References

[1] V. Antinyan, M. Staron, W. Meding, A. Henriksson, J. Hansson, and A. Sandberg, "Defining Technical Risks in Software Development," in *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on*, 2014, pp. 66-71.

[2] M. I. Kamata and T. Tamai, "How does requirements quality relate to project success or failure?," in *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, 2007, pp. 69-78.

[3] E. Knauss and C. El Boustani, "Assessing the quality of software requirements specifications," in *International Requirements Engineering, 2008. RE'08. 16th IEEE*, 2008, pp. 341-342.

[4] D. Damian and J. Chisan, "An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management," *Software Engineering, IEEE Transactions on,* vol. 32, pp. 433-453, 2006.

[5] M. Kauppinen, M. Vartiainen, J. Kontio, S. Kujala, and R. Sulonen, "Implementing requirements engineering processes throughout organizations: success factors and challenges," *Information and Software Technology,* vol. 46, pp. 937-953, 2004.

[6] S. Jacobs, "Introducing measurable quality requirements: a case study," in *Requirements Engineering, 1999. Proceedings. IEEE International Symposium on*, 1999, pp. 172-179.

[7] H. Mat Jani and A. Tariqul Islam, "A framework of software requirements quality analysis system using case-based reasoning and Neural Network," in *Information Science and Service Science and Data Mining (ISSDM), 2012 6th International Conference on New Trends in*, 2012, pp. 152-157.

[8] C. Patel and M. Ramachandran, "Story card maturity model (SMM): a process improvement framework for agile requirements engineering practices," *Journal of Software,* vol. 4, pp. 422-435, 2009.

[9] S. Beecham, T. Hall, and A. Rainer, "Defining a requirements process improvement model," *Software Quality Journal,* vol. 13, pp. 247-279, 2005.

[10] B. H. Cheng and J. M. Atlee, "Research directions in requirements engineering," in *2007 Future of Software Engineering*, 2007, pp. 285-303.

[11] B. Regnell, R. B. Svensson, and K. Wnuk, "Can we beat the complexity of very large-scale requirements engineering?," in *Requirements Engineering: Foundation for Software Quality*, ed: Springer, 2008, pp. 123-128.

[12] K. Wnuk, B. Regnell, and C. Schrewelius, "Architecting and coordinating thousands of requirements–an industrial case study," in *Requirements Engineering: Foundation for Software Quality*, ed: Springer, 2009, pp. 118-123.

[13] A. Van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in *Proceedings of the 22nd international conference on Software engineering*, 2000, pp. 5-19.

[14] N. P. Napier, L. Mathiassen, and R. D. Johnson, "Combining perceptions and prescriptions in requirements engineering process assessment: an industrial case study," *Software Engineering, IEEE Transactions on,* vol. 35, pp. 593-606, 2009.

[15] J. Bosch, *Continuous Software Engineering*: Springer, 2014.

[16] G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado, and V. Moreno, "A framework to measure and improve the quality of textual requirements," *Requirements Engineering,* vol. 18, pp. 25-41, 2013.

[17] L. E. Hyatt and L. H. Rosenberg, "Software metrics program for risk assessment," *Acta astronautica,* vol. 40, pp. 223-233, 1997.

[18] B. Gleich, O. Creighton, and L. Kof, "Ambiguity detection: Towards a tool explaining ambiguity sources," in *Requirements Engineering: Foundation for Software Quality*, ed: Springer, 2010, pp. 218-232.

[19] A. Geraci, F. Katki, L. McMonegal, B. Meyer, and H. Porteous, "IEEE Standard Computer Dictionary," *A Compilation of IEEE Standard Computer Glossaries. IEEE Std,* vol. 610, 1991.

[20] H. Zuse, "Software complexity," *NY, USA: Walter de Cruyter,* 1991.

[21] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*: CRC Press, 2014.

[22] V. Basili, "Qualitative software complexity models: A summary," *Tutorial on models and methods for software management and engineering,* 1980.

[23] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *Software Engineering, IEEE Transactions on,* vol. 22, pp. 68-86, 1996.

[24] E. Rechtin and M. W. Maier, *The art of systems architecting*: CRC Press, 2010.

[25] G. I. Susman and R. D. Evered, "An assessment of the scientific merits of action research," *Administrative science quarterly,* pp. 582-603, 1978.

[26] R. L. Baskerville, "Investigating information systems with action research," *Communications of the AIS,* vol. 2, p. 4, 1999.

[27] R. L. Baskerville and A. T. Wood-Harper, "A critical perspective on action research as a method for information systems research," *Journal of Information Technology,* vol. 11, pp. 235-246, 1996.

[28] P. Checkland and S. Holwell, "Action research: its nature and validity," *Systemic Practice and Action Research,* vol. 11, pp. 9-21, 1998.

[29] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh*, et al.*, "Identifying and measuring quality in a software requirements specification," in *Software Metrics Symposium, 1993. Proceedings., First International*, 1993, pp. 141-152.

[30] R. Vlas and W. N. Robinson, "A rule-based natural language technique for requirements discovery and classification in open-source software development projects," in *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, 2011, pp. 1-10.

[31] C. Huertas and R. Juárez-Ramírez, "NLARE, a natural language processing tool for automatic requirements evaluation," in *Proceedings of the CUBE International Information Technology Conference*, 2012, pp. 371-378.

[32] J. Kasser, W. Scott, X.-L. Tran, and S. Nesterov, "A proposed research programme for determining a metric for a good requirement," in *The Conference on Systems Engineering Research*, 2006, p. 1.

[33] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An automatic quality evaluation for natural language requirements," in *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ*, 2001, pp. 4-5.

[34] E. Parra, C. Dimou, J. Llorens, V. Moreno, and A. Fraga, "A methodology for the classification of quality of requirements using machine learning techniques," *Information and Software Technology,* vol. 67, pp. 180-195, 2015.