Testing Natural Language Application Recognition Accuracy and Habitability

TR 29.3297

March 20, 2000

James R. Lewis

Speech Development and Customization

West Palm Beach, Florida

Abstract

This document describes a strategy for assessing recognition accuracy and habitability for applications using spoken natural language user interfaces. The purpose of habitability testing is to measure the extent to which the grammar accepts the utterances that users make when speaking to the system. The purpose of recognition accuracy testing is to measure how accurately the system can interpret legal utterances. Both recognition accuracy and habitability are important components of system usability for spoken natural language user interfaces. The best test strategy uses multiple test sets with complementary evaluative properties.

ITIRC Keywords

Natural language application
NLU
BNF grammar
Natural command grammar
Minimal test set
Accuracy assessment
Habitability assessment
Complementary test sets

Contents

Executive Summary

This document describes a strategy for assessing habitability and recognition accuracy for applications using spoken natural language user interfaces. The purpose of habitability testing is to measure the extent to which the grammar accepts the utterances that users are likely to make when speaking to the system. The purpose of recognition accuracy testing is to measure how accurately the system can interpret legal spoken utterances. Both habitability and recognition accuracy are important components of system usability for spoken natural language user interfaces[1].

After analyzing various approaches to testing habitability and recognition accuracy, it appears that the best order for testing these system properties is habitability first, followed by recognition accuracy testing. The reason for this is that the process of testing habitability has as an output a set of user-generated phrases. Habitability testing reveals which of these phrases the system is capable of recognizing and which are not legal utterances for the system. Developers can use this information to broaden the language model of the system, increasing its habitability.

After completing the development activities that broaden the system's language model, the set of legal user-generated phrases becomes an excellent set for testing the system's recognition accuracy. However, there is no guarantee that the set of user-generated phrases effectively exercises the boundaries of the system's capabilities.

To address this, it is necessary to develop a complementary set of test phrases based on analysis of the system's specified capabilities. There are a number of ways to approach this development, with the best way depending on the available system development tools.

Testing with these two complementary sets (user-generated for a set of natural test phrases and developer-generated for a set of comprehensive test phrases) should provide reasonable coverage for an efficient assessment of a system's recognition accuracy[2].

[1] They are necessary but not sufficient for guaranteeing system usability. The creation of usable systems requires other, more standard usability development activities as well, described in a number of texts on the topic of usability design and evaluation.
[2] One topic not addressed in this report is the level of acceptable habitability or recognition accuracy. The acceptable level of accuracy for an application depends on its intended use (LaLomia, 1993), and this almost certainly applies to habitability as well.

Testing Habitability

Habitability

Both natural command[3] and natural language[4] applications can have habitability problems. Habitability is the technical term for the ease, naturalness and effectiveness with which users can use language to express themselves when working with a natural language system (Watt, 1968). There are four domains within which users must stay when speaking to a natural language application: conceptual (you can't ask a traffic report system about stock quotes), functional (the system might handle requests that contain multiple tokens[5], or might require users to create single-token phrases), syntactic (the system might allow many or few paraphrases for commands), and lexical (the system might allow many or few synonyms for the words in the application).

"A natural language system must be made habitable in all four domains because it will be difficult for users to learn which domain is violated when the system rejects an expression" (Ogden and Bernick, 1997, p. 139). For example, consider system rejection of the phrase, "Provide today's arrival times and destinations for Delta Flight 134." If the user was talking to a stock quote application, then the violation was in the conceptual domain. If the system could provide arrival times or destinations, but not both from a single request, then the violation was in the functional domain. If the system would have understood the request in a different syntactic form ("What are today's arrival times and destinations for Delta Flight 134?), then the violation was in the syntactic domain. Finally, if the system would have understood the command if the user had substituted 'list' for 'provide', then the violation was in the lexical domain.

Evaluation Issues

Ogden and Bernick (1997) describe several habitability evaluation issues, including user selection, training, task generation, and task presentation.

As with any user evaluation, the participants should be representative of the intended user population. Participants should receive training in the use of the system that is representative of the training that will be available to the intended user population.

Tasks can be either user- or tester-generated, but tester-generated tasks based on the functional specification have greater assurance of more comprehensive functional coverage than user-generated tasks. On the other hand, tester-generated tasks will not

[3] A natural command application allows users to use synonyms and multiple ways of phrasing commands, but uses a structured grammar (such as BNF) which has been written by a human developer to define the utterances that the system can recognize.
[4] A natural language application also allows users to use synonyms and multiple ways of phrasing commands. Rather than being built by a human developer, the system uses statistical analysis of relevant samples of text to infer what users are likely to say to it.
[5] A token is the smallest unit of meaningful information in an utterance. For example, consider the sentence, "Oh, I think I might like to travel to Tampa next Saturday.", spoken to a system for booking air travel. Even though the sentence contains twelve words, it only contains two tokens – 'Tampa' (destination) and 'next Saturday' (departure date).

address conceptual habitability because the tester typically knows the conceptual limitations of the system.

If at all possible, the tester should present tasks[6] to the user with tables or graphs, asking the participant to say what he or she would say to the system to get the information needed to put information into the cells of the table or to complete the task indicated on the graph[7]. Figure 1 shows some examples of these types of task representations.

Figure 1. Examples of Tabular and Graphical Task Representation

---

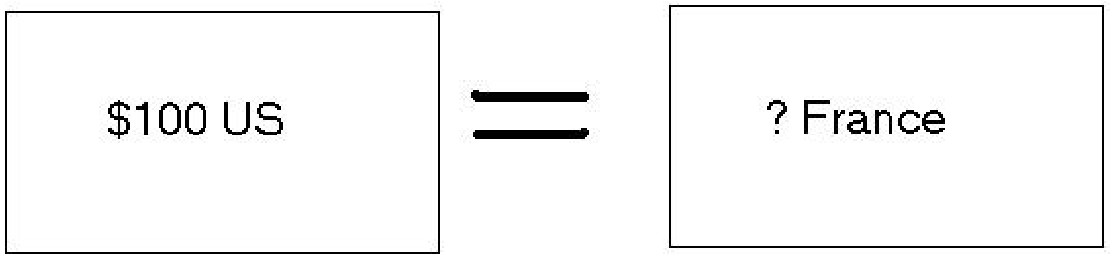
TABULAR – AIRLINE FLIGHTS

Carrier	**Flight No.**	**Departure City**	**Departure Time**	**Arrival City**	**Arrival Time**
Delta	123	Miami	12:00 PM	Atlanta	
	876	Dallas	8:00 AM	Seattle	11:00 AM

Potential user input

What time does Delta flight 123 arrive in Atlanta?
Tell me the airlines that leave Dallas for Seattle around 8:00 AM?

GRAPHICAL – CURRENCY CONVERSION



Potential user input

How many French francs can you get for one hundred US dollars?
Tell me the current rate for converting US dollars to French francs, please.

The advantage of this type of task representation is that it induces less linguistic bias to participants than writing the task in words (although it is often impossible to construct even these types of scenarios without any words). For a word-processor, the tester could

[6] Tasks should be consistent with the application's conceptual and functional domains.
[7] Could this approach adversely affect the validity of the phrases generated by participants when the application will not have a graphical user interface (GUI)? Perhaps, but it seems unlikely that this effect would have more adverse effect than the lexical and structural cues that participants would pick up from scenarios composed only of words.

give the participant a marked-up copy for a document already on the system with the instruction to get the file and make all the indicated changes.

If it isn't possible to represent the task in this way, then an alternative approach is to present the task as a larger problem that requires several steps to complete. (For example, "Use this system to send a message to Bill Smith asking him if he would be free for lunch next Friday. Be sure to send it immediately."[8])

Although not absolutely required, it is usually best to conduct this type of user study with a prototype of the application (as opposed to simply presenting the scenarios to the user and recording what they say). The prototype can be either a working prototype or a Wizard of Oz (WOZ) prototype[9]. The purpose of a WOZ prototype is to find out via simulated speech-enablement of the application what people are likely to say to control the application. In a WOZ study, people who are representative of the target product audience attempt to use speech to control a simulation of the application. This procedure requires a setup in which the users can see the application on a display (or, in the case of developing telephone or other displayless interfaces, can hear the application). What the users cannot see is the "wizard behind the curtain" – a person who can hear what the users say, and controls the application appropriately to produce the desired effect. If the user's intention is not clear, then the wizard should be able to display to the user a request to try again using, if possible, the same feedback mechanism that the working application would present. Doing a WOZ study requires a functional specification of the application, but does not require a working prototype. Regardless of the prototyping method, it is a good idea to record the participants' speech for transcription and potentially for use as input in an evaluation of recognition accuracy.

The more people participating in the prototype study, the better, up to the point that additional participants provide few additional phrases. There are no published experiments on the effect of running different numbers of participants on the breadth of expressions produced, but similar work in problem-discovery usability studies (Lewis, 1994) suggests that six to ten participants would be a reasonable sample size for planning purposes. As we gain more experience doing this type of work, we will be able to make stronger statements regarding adequate sample sizes. If the application's audience includes very young or old users, then the sample should include participants whose ages span the relevant range. This also applies to applications intended for use by people speaking different dialects, or who have other relevant cultural or experiential differences. The more of these types of user characteristics that it is important to sample, the larger the sample size will need to be.

---

[8] Avoid atomic descriptions of tasks, such as, "Create a new message. Address it to Bill Smith. Dictate 'Are you free for lunch next Friday?' Send the message immediately."

[9] Scientists at IBM's T.J. Watson Research Institute, such as John Gould and Jeff Kelley, pioneered this technique in the early 1980s. For example, see Gould, J. D., Conti, J., and Hovanyecz, T. (1981) or Kelley, (1985). For more recent work using WOZ to develop speech systems, see Bernsen, N. O., Dybkjaer, H., and Dybkjaer, L. (1996).

Using the Output of Habitability Testing

After gathering the participants' utterances obtained during habitability testing, the next step is to determine which utterances the application is inherently capable of recognizing. The precise way to do this depends on the tools available in the development tools for the system. For systems with formal grammars, the toolkits usually provide a tool for checking to see if a phrase is in-grammar. Other natural language toolkits could provide tools for checking to see that all the words are in the language model, or could even check to see that the system parses the utterances and correctly extracts the tokens. Developers can use the output of this analysis in two ways:

- Alter the system so it can correctly interpret more of the utterances
- Use the correctly interpreted utterances as a phrase set for testing recognition accuracy (described in more detail in the following section)

Testing Recognition Accuracy

There are many aspects of natural language processing that require evaluation (Spark Jones and Galliers, 1996), with many of these associated with the quality of linguistic analysis associated with parsing and extraction of meaning. It is important to keep in mind that natural language processing systems might or might not involve speech recognition. Recognition accuracy testing is only relevant for natural language processing systems that include speech recognition.

Three questions to answer before conducting a recognition accuracy test are:

- How many phrases should you collect from each speaker[10]?
- How do you select the phrases?
- How many speakers should participate in the test?

The answer to the third question depends on measurement variability and required measurement precision (Walpole, 1976). Previous analyses (Lewis, 1997) have indicated that eight speakers will typically yield measurements with acceptable precision, although we have successfully used as few as four and as many as twelve speakers in various studies. Because accuracy can vary as a function of gender, tests should include both male and female speakers (in equal numbers, if possible, for the best experimental design). If the application's audience includes very young or old users, then the sample should include speakers whose ages span the relevant range. This also applies to applications intended for use by people speaking different dialects, or who have other relevant cultural or experiential differences. The more of these types of user characteristics that it is important to sample, the larger the sample size will need to be. Testers need to balance the need for adequate coverage of user characteristics with the available resources.

The first two questions apply to the development of the set of test phrases, with four potential strategies for this development.

Strategy 1: Use Application Functions to Guide Test Set Development

I recently developed a strategy for answering these questions (number of phrases and phrase content) when conducting accuracy tests for natural command grammars (Lewis, 2000). For natural command grammars (finite-state grammars that allow natural-language-like flexibility in the number of allowable expressions per function) the number

[10] One might also ask how many tries each speaker should have with each phrase, and whether or not it is necessary to have each person speak exactly the same phrases. For evaluations using recorded phrases, it only makes sense to have each person speak each phrase once. For live evaluations, there might be some interest in seeing if users can adapt their speech patterns to increase accuracy after a misrecognition, but the correlation between one-try and multiple-tries accuracy will necessarily be high. Whether or not the test should allow multiple tries depends on the purpose of the evaluation, although in most cases allowing multiple tries will cause tests to take longer with little increase in information. To be able to confidently identify high-failure phrases after the test has finished, the best strategy is to have each person speak each test phrase. The only time a tester wouldn't do this is if there were just too many phrases that needed testing and too little time to have each person speak each phrase.

of valid commands is for all practical purposes infinite, precluding exhaustive testing. (In this regard, natural command grammars are similar to natural language applications.) It is possible to use the natural command grammar to produce the smallest possible set of test cases that can comprehensively test the grammar with regard to verifying correct output to a translation rule interpreter or other parsing system (in other words, based on functions). This minimal test set is also appropriate for testing accuracy because it generates the full range of the types of phrases that are legal in the grammar and uses all of the words in the grammar. See Lewis (2000) for a method of constructing this minimal test set.

If the natural command grammar (or natural language system) includes large lists (such as stocks listed in stock exchanges), then it is necessary to limit the phrases using the members of these lists to a manageable number. For example, there are roughly 8,000 airports in the world, but almost all of the world's airport traffic moves through about 600 of those airports. It seems reasonable to limit accuracy testing to the 600 airports that account for over 99% of the world's airport traffic[11].

A key difference between natural command and natural language systems is that a developer constructs a natural command grammar by hand, but uses statistical processes to produce a natural language system from the analysis of a collection (corpus) of domain-relevant phrases. The designed structure of a natural command grammar provides the information needed to develop the minimal test set. This is especially easy if the grammar includes translation rules (see Lewis, 2000, for an example of a grammar with translation rules), although the method does not require translation rules. A natural language application, however, might not contain a structure like a finite-state grammar, so it is necessary to develop an alternative way to determine a phrase set similar to the minimal test set of a natural command grammar.

One way to do this is to use the functions of the application as the basis for the generation of test phrases (similar to the strategy of basing a test set on a natural command grammar's translation rules – which translate phrases into their functional equivalents for an application using a natural command grammar). Even though the number of legal phrases for a natural language application might be essentially infinite, the number of functions is certainly finite and most likely reasonably small – probably not more than a few hundred at most, and possibly far less. To achieve sufficient coverage of legal phrases, there should be enough variants of the test phrases to include all the members of short lists (for example, colors or single-digit numbers), known synonyms for the identified objects and actions of the interface (for example, if 'delete' is a valid action in the interface, develop phrases that include synonyms such as 'remove' and 'erase'), and the key items from long lists (such as airport names). In other words, it isn't necessary (or feasible) to test all possible combinations, but it makes sense to attempt to test all possible structures.

[11] Even 80 or 90% coverage would be fine for most evaluations. The idea is that in any large list, a relatively small number of items will account for most cases. Testers need to strike a balance between the number of items they have the time to test and the amount of coverage they desire.

The next step is to process a text file containing the test phrases to ensure that the recognizer can recognize it (in other words, the test phrases do not contain any words or structures that the system is inherently incapable of recognizing). The tester needs to rewrite all phrases that fail this test and then test the rewritten phrases until all phrases in the test set are valid in the context of the existing application. If the natural language development system produces a finite-state grammar for the application, then the toolkit will probably contain a tool for testing the legality of a given phrase in a given grammar[12]. If, instead, it produces a vocabulary and language model (similar to that of a dictation system), it should be possible to use a tool like the ViaVoice[13] Vocabulary Expander to ensure that all of the words in the set of test phrases are in the system's vocabulary.

This strategy for the development of a test phrase set doesn't have a requirement for a set number of test phrases, but allows the number to vary depending on the functional breadth of the application, the likely use of synonyms, and the use of small and large lists of items. Thus, the coverage of the test set is excellent while keeping the set as small as possible. (If this procedure produces fewer than 100 phrases, then create two versions of each phrase that can take more than one form to obtain a test set that has reasonable measurement sensitivity – at least 100 phrases[14].) The resulting set of test phrases has important potential use beyond recognition accuracy testing. For example:

- Programmers can use the set to test an application's parsing accuracy.
- Information developers can use the set to guide the selection of examples for user's guides and online help, focusing on promoting the use of phrases known to have high recognition accuracy.

A disadvantage of the strategy is that no current automated tool can produce the set[15], and it is laborious and time-consuming to produce by hand. To extract a test set using this method requires documentation of the application, either by a finite-state grammar or by a comprehensive functional specification.

Strategy 2: Use the Output of Habitability Testing

This approach has the advantage of providing reasonable assurance that the test phrases are representative of the types of things users are likely to say when using the application – something that no other strategy can provide. The disadvantage is that there is no guarantee that collecting phrases from a relatively small group of participants will provide the same extent of coverage as the minimal test set of Strategy 1. Note that one would only use the phrases from habitability testing that are legal utterances in the application.

[12] Most toolkits for grammar-based systems contain a tool for testing the legality of phrases (for example, fsgtest.exe in the IBM ViaVoice Developer's Toolkit).
[13] ViaVoice is a trademark or registered trademark of International Business Machines Corp.
[14] If you have information about the relative frequency of use of the functions, you can use this as a guide to oversampling functions to increase the number of phrases in the test set (or, in habitability testing, the number of scenarios devoted to eliciting utterances for the different functions).
[15] Although such a tool should be possible to build.

Strategy 3: Random Generation of Test Set Phrases

If the developer's toolkit for natural language applications contains a tool for randomly generating grammatical phrases (for example, a tool like fsgenum.exe in the IBM ViaVoice Developer's Toolkit), then an alternate strategy for developing a test set would be to generate about 1000 phrases to use for recognition accuracy testing. The number of phrases to generate for the test set depends on the functional breadth of the application. For testing natural command grammars, we have routinely used test sets that exceed a total of 1000 phrases. For other recognition accuracy experiments, we have sometimes used as few as 100 phrases in the test set.

A key advantage of this strategy is that it is very easy to generate the test phrases with this strategy. A disadvantage of the strategy is that there is no guarantee that the phrases generated will provide an efficient evaluation of potential acoustic confusability, especially among the members of item and synonym lists. Also, the test set might not provide adequate functional, syntactic or lexical coverage[16]. Finally, if the output of the process used to generate the natural language application is a vocabulary and language model rather than a finite-state grammar, the toolkit is not likely to have any way of automatically producing meaningful, grammatical test phrases.

Strategy 4: Use Phrases from the Natural Language Application's Training Set

It might be tempting to draw the test phrases from the materials used to generate (train) the natural language application. Using test phrases drawn from the source material, though, would very likely lead to an overestimation of the accuracy of the system due to the unrealistically close match between the words in the phrases and the content of the language model (or finite-state grammar) produced by the natural application development system[17].

Using the Output of Recognition Testing

Some phrases will have better recognition than other phrases. Analysis of the phrases with relatively high-failure rates across participants can lead to changes to improve system accuracy. These changes can include:

- Modifying the grammar/language model of the system
- Adding additional pronunciations for frequently-misrecognized words in the system

[16] In fact, the more common use of randomly generated phrases is to identify awkward, unintended or incorrect phrases that a grammar/language model under development allows. The developer uses this technique to find out what shouldn't be in the grammar/language model so he or she can take steps to eliminate them from the application.

[17] It is a general statistical principle that one should not use the data from which a statistical model was generated to assess the effectiveness or quality of the model. Doing so will always make the model seem to be better than it really is.

Summary

Habitability Testing

Habitability testing requires a prototype (either working or Wizard of Oz). The study should include six to ten participants, with the participants completing tester-generated tasks (because tester-generated tasks are more likely than user-generated tasks to provide adequate functional coverage). The best way to represent the task (when possible) is as missing cells in a table or as a graph.

Accuracy Testing

Strategy 1 (development of a minimal test set) is the best strategy for recognition accuracy assessment. It is the only approach that guarantees complete functional coverage, along with complete coverage of legal vocabulary and phrase structure. To apply this strategy, the test developer must have access to either a comprehensive functional specification describing the application or the output of the natural language development process must be a finite-state grammar. An advantage of this strategy is that users do not have to be familiar with the application.

Strategy 2 (using the output of habitability testing) is a reasonable method to employ for situations in which the test developer doesn't have the required documentation to produce a minimal test set. As long as the test developer has some specification of the functions of the application, it should be possible to construct scenarios that will elicit realistic test phrases from the participants in the study. The more comprehensive the specification, the better, but it is possible to start development of the set of test phrases with fairly limited documentation. A disadvantage of this strategy is that user must exist who are familiar with the application (either by previous use or by pre-test training).

Strategy 3 (random generation of test phrases) is a less effective method (although it, like Strategy 1, does not require a set of users familiar with the application). A serious downside is that many natural language toolkits will not even contain a tool that can do this. If the tool exists, then the phrases it produces will not have the desirable properties of either of the other two strategies (comprehensive coverage for Strategy 1, naturalness for Strategy 2).

Using Strategy 4 (using phrases from the training set used to product the natural language application) will lead to an overestimation of the accuracy of the system. Developers should avoid this strategy.

Regardless of which strategy a tester uses to develop the set of test phrases, to ensure adequate precision of measurement, the test should:

- Include six to twelve speakers with equal (or as equal as possible) numbers of male and female speakers (and addressing other coverage issues as required).
- Have at least 100 phrases in the test set.

Putting It Together

It generally makes sense to assess habitability before assessing recognition accuracy. This allows developers to take advantage of the output of the habitability study to refine the application (letting the user-generated phrases indicate where the existing application needs to become more flexible in the functional, syntactic, and lexical domains). Once the developers have extended the application's flexibility, the final habitability score is the percentage of user-generated phrases that are legal expressions in the application. These legal expressions can become one of the test sets for assessing recognition accuracy (Strategy 2).

The disadvantage of a Strategy 2 test set is that it might not examine the full range of functions, syntaxes, and words that are legal in the application. Therefore, if it is possible, the tester should also develop a Strategy 1 test set, and use both test sets to assess the application's recognition accuracy. This is the best approach because the two types of test sets have complementary strengths and weaknesses.

If the application includes the use of large lists (for example, stock names), then the tester should develop test sets for each large list, testing only the most important members of the list to determine if any pairs of important members have high acoustic confusability and if the system confuses any less important members with important members.

This approach, using multiple test sets with different and complementary evaluative properties, should provide the test coverage necessary to obtain a comprehensive assessment of both recognition accuracy and habitability for natural command and natural language applications[18].

[18] In reviewing this paper, Alan Happ (personal communication, February 7, 2000) pointed out that it "struck me that the paper nearly talks about internal and external validity issues. The speech application is internally valid if it recognizes words it is programmed to recognize, and it is externally valid if its habitability allows users to effectively interact with the application in a natural way."

References

Bernsen, N. O., Dybkjaer, H., and Dybkjaer, L. (1996). Cooperativity in human-machine and human-human spoken dialogue. *Discourse Processes*, *21*, 213-236.

Gould, J. D., Conti, J., and Hovanyecz, T. (1981). *Composing letters with a simulated listening typewriter* (Tech. Report RC 9119). Yorktown, NY: International Business Machines Corp.

Kelley, J. F. (1985). *CAL -- A natural language program developed with the OZ paradigm: Implications for supercomputing systems* (Tech. Report RC 11324). Yorktown, NY: International Business Machines Corp.

LaLomia, M. J. (1993). *An evaluation of user acceptance of handwriting recognition accuracy* (Tech. Report 54.761). Boca Raton, FL: International Business Machines Corp.

Lewis, J. R. (1994). Sample sizes for usability studies: Additional considerations. *Human Factors, 36,* 368-378.

Lewis, J. R. (1997). *A general plan for conducting human factors studies of competitive speech dictation accuracy and throughput* (Tech. Report 29.2246). Raleigh, NC: International Business Machines Corp.

Lewis, J. R. (2000). *Developing minimal comprehensive sets of test cases for natural command grammars* (Tech. Report 29.3272). Raleigh, NC: International Business Machines Corp.

Ogden, W. C., and Bernick, P. (1997). Using natural language interfaces. In M. Helander, T. K. Landauer, and P. Prabhu (eds.), *Handbook of Human-Computer Interaction* (pp. 137-161). Amsterdam, Netherlands: Elsevier.

Sparck Jones, K., and Galliers, J. R. (1996). *Evaluating natural language processing systems*. Berlin, Germany: Springer.

Walpole, R. E. (1976). *Elementary statistical concepts*. New York, NY: Macmillan.

Watt, W. C. (1968). Habitability. *American Documentation*, July, 338-351.