Issues in Test Suite Design

Lorna Balkan, Siety Meijer, Doug Arnold,
Eva Dauphin, Dominique Estival, Kirsten Falkedal,
Sabine Lehmann, Klaus Netter, Sylvie Regnier-Prost.

Report to LRE 62-089 (D-WP2.1)

Test Suites for Natural Language Processing (TSNLP)

University of Essex

July 1994

Summary

Title	Test Suite Design — Guidelines and Methodology
Project	LRE-62-089
Document ID	D-WP2.1
WP	Test Suite Design
Date	July 1994
Principal Authors	Lorna Balkan and Siety Meijer
Authors	Lorna Balkan, Siety Meijer, Doug Arnold, Eva Dauphin, Dominique Estival, Kirsten Falkedal, Sabine Lehmann, Klaus Netter, Sylvie Regnier-Prost
Reviewers	The consortium
Status	Intermediate Report; will undergo revisions

Contents

1 Introduction **1**

2 Literature Review **1**

2.1 Introduction . . . 1

2.2 Galliers and Sparck-Jones . . . 1

2.3 EAGLES . . . 4

2.4 Neal Montgomery Method . . . 9

2.5 The Sourcebook Project . . . 10

2.6 Heid and Hildenbrand . . . 12

2.7 Survey for *The Editor's Assistant* . . . 13

2.8 Aerospatiale methodology . . . 15

2.9 DITO . . . 17

2.10 Gambäck . . . 18

2.11 Lehrberger and Bourbeau . . . 20

2.12 Discussion . . . 22

3 Evaluation type **23**

3.1 The conditions of testing . . . 24

3.2 The characteristics of the test items . . . 25

3.3 The testing procedure . . . 26

4 System/application/component type **26**

4.1 System-level test suites . . . 26

4.2 Application level test suites . . . 27

4.3 Component-level test suites . . . 27

4.4 General purpose test suites . . . 28

4.5 Discussion . . . 28

5 System input **29**

5.1 Text type . . . 29

5.2 Authorship . . . 29

5.3 Modality . . . 29

5.4 Discussion . . . 30

6 System architecture and test suite design **31**

6.1 Statistical Based Systems . . . 33

6.1.1 Current Techniques in Testing Statistical Based Systems 33
6.1.2 Test Suites and Statistical Systems 34

7 System output **35**

8 User type **38**

9 Comparing Test Suites with Test Corpora **39**
9.1 General Features 39
9.2 Coverage 39
9.3 Presentation 40
9.4 Re-usability 40
9.4.1 Discussion 41

10 Limitations of Test Suites and their Effects on Test Suite Design **41**
10.1 Evaluation type 42
10.2 System/Application/Component Type 43

Bibliography **45**

Issues in Test Suite Design

1 Introduction

This document provides the background material for the first part of the WP2.1 deliverable: "Test Suite Design: Guidelines and Methodology". It begins with a literature review, which introduces some design issues in test suite construction. These design issues are then discussed in greater detail in the sections 3 to 8 and include:

- Evaluation type
- System/application/component type
- System input
- System architecture
- System output
- User type

There is no suggestion here that this list is definitive, but the above represent at least some important factors in test suite design.

Section 9 discusses the relationship between test suites and test corpora. Finally, we consider the limitations of test suites and summarize all these issues in section 10,

2 Literature Review

2.1 Introduction

Here we provide a selective overview of what has been written about test suites in the literature. The first two sections (2.2 and 2.3) present two prominent general evaluation frameworks. We concentrate on the evaluation of linguistic performance. The evaluation framework in the remaining sections is in some cases less elaborated, but the sections have been included because they represent interesting views on the construction of test suites.

2.2 Galliers and Sparck-Jones

Galliers and Sparck-Jones (GS)([10]) describe an evaluation framework centred on the notion of performance factors.

They claim that a good way, if not the only proper way to approach evaluation is to look first at the types of task and of application, to see what these imply for language processing and hence for NLP systems, rather than to start with some notion of what a language processor per se ought to be able to do and then attempt to derive one's evaluation techniques from this.

They summarise their evaluation framework in two recommendations:

1. Unpack the evaluation by working systematically through the series of relevant questions (see list in Table 1 below), pushing into the necessary detail and decomposing the evaluation subject in corresponding detail for proper answers.

2. Envisage the evaluation from the start as a programme of coherently-related component evaluations based on a performance factor grid.

	EVALUATION REMIT
to establish:	
	motivation - why evaluate
	perspective - task/financial/administrative/scientific ...
	interest - developer/funder ...
	consumer - manager/user/researcher ...
	goal - what discover
	kind - investigation/experiment
	type - blackbox/glassbox
	form (of yardstick) - ideal/attainable/examplar/given/judged
	style - suggestive/indicative/exhaustive
	mode - quantitative/qualitative/hybrid
	EVALUATION DESIGN
to identify:	
	subject's end - what's subject for
	subject's context - what's it in
	subject's constitution - what's it of
to determine:	
	performance factors
	environment variable
	"system" parameters
	performance criteria
	performance measures
	application methods
	evaluation data
	evaluation procedure

Table 1: FRAMEWORK QUESTIONS FOR EVALUATION SCENARIO

It must be stressed that there are dependencies between these questions and that it may be impractical to act on some decision because, for example, the required data is unobtainable or too costly to obtain. Answers to some questions may mean that answers to earlier ones have to be revised.

The first few questions (relating to motivation, perspective, interest and consumer) are self-explanatory. The particular goal of evaluation will determine what it is the evaluation is intended to determine and will consequently determine the orientation of the evaluation, i.e. whether it is "intrinsic" (relating to a system's objective) or "extrinsic" (relating to a system's function). The distinction between a system's objective and its function is illustrated in the example of an NLP interface to a database management system. The objective is to answer data queries cooperatively, while the function is to meet data needs.

The distinction between blackbox and glassbox is that the former focuses on what a system does and is in principle possible without knowing anything about the inner workings of the system, while glass box evaluation attempts to look inside the system and to find ways of measuring how well the system does something, rather than simply whether or not it does it.

The answers to the first set of questions, on the *evaluation remit*, lead to the second set of questions on *evaluation design*. These fall into three classes:

- Definition of the subject of evaluation

  For systems, the subject's ends are, their objective or function. The subject's context includes animate and inanimate elements, e.g. who the users are. It is important to identify the wider setting in which the system is used - for instance whether it is interactive or not. Interactive systems may impose different NLP requirements, like those characteristic of dialogue. The subject's constitution refers to the system's components.

  Systems may not be exclusively devoted to NLP, so it is also necessary to determine what the system tasks are, and in many cases to decompose systems to separate out the language subsystems and components. It is claimed that the crucial issue in evaluating the NLP component when the system's task is not itself exclusively devoted to NLP is in establishing the division between l-system (that part of a system that engages in language processing) from the non-language system (the n-system). It is further claimed that if the separation between the two is weak, it is hard to establish and evaluate l-system performance independent of n-system performance.

  The characterisation of ends is needed on the one hand to validate the remit, and on the other hand to validate the evaluation criteria, since you need to know a subject's objective in order to choose sensible criteria, measures and methods (see below).

- Performance factors

  Performance factors can be answered once characterisation of context and constitution questions have been answered. The distinction between variables and parameters is between external environment factors embodying givens from the system designer's point of view (variables), and internal system factors embodying the designers choices (parameters), which are together responsible for the system's effects.

  By way of example, a selective dissemination of information (SDI) system is given. The system is given document titles and abstracts and it uses standing interest profiles for staff members to search titles plus abstracts for matching word stems. It ranks output in descending order of number of matching terms and circulates titles and abstracts for the top twenty documents to each user once a week. Environment variables include the subject field of the documents, the journals from which material is taken, the size of the abstracts, etc. The system's parameters include the choice of text area to search, various parameters related to the nature of the indexing including term form and grounds for selection, the matching coefficient used, the choice and style of output, etc.

- Criteria, Measures and Methods

  Different categories of criteria are distinguished:

  - intrinsic versus extrinsic criteria

    Intrinsic criteria are those relating to a system's objective, extrinsic criteria are those relating to its function. For example, for an MT system, intrinsic criteria could be those applying to the quality of translation, and extrinsic criteria those appyling to the ease with which post-editors could establish these.

  - wide-bound versus narrow-bound criteria

    In awide bound evaluation, performance criteria are intentionally adopted which make no reference to internal system design.

  Criteria are translated into measures which are "very specific criteria in disguise". Methods are defined for the application of the chosen measure and include all aspects of experimental design.

  By way of illustration, an MT system is presented. Criteria include *translation*, which is decomposed into the measures of *intelligibility*, *style* and *accuracy*. A method for assessing intelligibility might be a readability scale, and a method for assessing style subjective grading.

- Choice of evaluation data

  This relates to the nature and status of the data and covers questions about its representativeness and legitimacy.

Test suites

No precise methodology is defined for producing test suites, but a number of observations about test data (including test corpora and test suites) is offered about their representativeness and legitimacy. Representativeness is a major issue for all evaluation data, not just test suites: what is the data meant to represent and how is it known that it does indeed represent this?

As [10] point out, there are two issues at stake with test data:

- Validity
- Appropriateness

Test data is *valid* if it represents what it is supposed to represent. Even with data drawn from the real world, it is important to be clear about the nature of the variables it instantiates, as well as whether it does instantiate them in a statistically valid way.

The question of *appropriateness* relates to whether the test data is appropriate as test data for the NLP system of interest.

Thus it is necessary both to fully characterise the test material in its own right and to show that it is appropriate material for the evaluation in which it is to be exploited.

2.3 EAGLES

The EC sponsored Expert Advisory Group on Language Engineering Standards (EAGLES) subgroup on NLP Evaluation and Assessment is engaged upon defining an adequacy evaluation methodology for NLP systems. The aim is to identify a set of typical needs which are potentially of interest to an end user or to a class of end users, and to define ways of measuring a system's adequacy with respect to those needs. An analogy can be made with consumer reports found in consumer association magazines, where a range of products are compared in terms of certain characteristics, and where the individual customer has to decide which characteristics are of importance to him and make his choice on that basis.

The design of an evaluation can be thought of as involving a series of steps, each successive step being guided by the previous steps:

- definition of relevant quality characteristics
- definition of attributes pertinent to each characteristic
- definition of measures able to provide values for each attribute
- definition of methods whereby the particular value for a particular system can be determined

Quality characteristics

A first step in designing an evaluation is determining what quality characteristics are relevant and how they should be refined. Quality characteristics are the basic characteristics of a system which are of potential interest to the evaluator. The ISO 9126 characteristics for quality assessment can be taken as an example:

- Maintainability
- Functionality
- Reliability
- Usability
- Efficiency
- Portability

Each of the quality characteristics can be further refined. "Functionality" for example, can be broken down to give, as ISO 9126 states "a set of attributes" relating to different functions the system is supposed to perform. As a simple example, the basic functions to be expected of a spelling checker might include:

- detection of mis-spelt words where the mis-spelling does not correspond to a legitimate word of the language
- proposal of plausible corrections

from which the following attributes can be derived:

- all mis-spelt words which do not correspond to a legal form of the language are detected
- the correct form of the mis-spelt word is among the corrections proposed

The values of attributes are arrived at in different ways:

- Through inspection
- Through testing

One means of ascertaining the level of error detection for grammar checkers, for example, is to devise a test, using test suites for example, from which a percentage of errors detected might be obtained as a value. The values of certain attributes (e.g. ease of use) might be more subjective.

For each attribute, its value is expressed in terms of a **measure**. One measure might be percentages, for example. Each attribute also has an associated **method** which is used to arrive at the value, expressed in the appropriate measure.

Test suites

One of the subgroups within the EAGLES Evaluation and Assessement subgroup is the writers' aids subgroup. This subgroup ([3]) is in the process of elaborating an evaluation methodology for grammar checkers. A system's functionality determines a set of quality characteristics, from which a set of **requirements** is defined. One requirement for grammar checkers for example might be that the system should specify the grammatical errors made by the writer in such a way that the end-user is able to correct them. The group is also in the process of defining a **task model** which is intended to clarify the parameters along which an understanding of the requirements might vary. The task model is in the process of being elaborated, but at present contains:

- A number of input models, corresponding to the set of subjects of the various actions of the task. Subjects include the writer and end-user.
- A number of output models or objects of the various actions in the task outline. Objects include the input text.
- Elements of the task model include:
  - A model of writer types
  - A model of end-user types
  - A model of unproofed text
  - A model of proofed text
  - A Taxonomy of Grammar Errors
  - A Taxonomy of advice types

The writer model for example, contains features that are relevant to the writing task, e.g. proficiency in the language, professional training.

The taxonomy of errors is used to derive the test suite. The error taxonomy can be derived in a number of ways. For example, it can be derived by a comparison of unproofed text (input text) and proofed text. Also, different classification schemes are possible. For example, errors may be classified according to their *source*. A taxonomy of this kind is presently being considered by the group, where each error type is mapped onto the transformations of the model of the proofed text at the analytical level appropriate to the error type:

- OCR errors:

  They are represented by typical letter or bigram substitutions on the end-product (literal output) of the proofed model.

- second language errors:

  Second language errors such as wrong prepositions in prepositional phrases are represented as lexeme substitutions or possibly as subcategorisation errors if they are closely linked to verb frames

- Concentration lapses:

  Concentration lapses or planning errors such as failed agreement between noun phrase determiner and head will be represented as the breaking of a constraint in the grammar of noun phrases

Another type of error taxonomy that has been considered within the group is that based on parts of speech. A partial view of this taxonomy is given below:

Word Formation Errors

Noun Phrase Errors

1. Modifier-NP head disagreement errors:
   *A dogs
   *Many dog
2. Phonetic adjacency disagreement errors:
   (a) where phonetic value follows initial letter:
       *An dog
       *A uncle
   (b) where sound of initial letter is non-standard:
       i. in lexical 'words':
          *An unicorn
          ?A hotel
       ii. in 'sounded out' abbreviations and contractions:
          *An U.N. observer
       iii. in 'pronounced' abbreviations and contractions:
          *A UNCED

3. Modifier combination errors:

Adjectival Phrase Errors

Prepositional Phrase Errors

Adverbial Phrase Errors

Verb Phrase Errors

1. Object marking errors
2. Errors in other verb frame role markers
   - Wrong preposition
   - Missing Preposition

Sentence Errors

1. Subject head/verb phrase head disagreements
   *The dogs runs.

One purpose of an error taxonomy is that it should allow an easy mapping onto a set of reportable **attributes**, which in the EAGLES model arise from the set of quality characteristics. Attributes also relate to the task model to the extent that the set chosen reflects the interests of the customer or user. (Where the customer is different from the user of the system, then a model for the customer must be added to the task model).

Generating test suites

The Writers' aids subgroup is also considering the question of how to generate test suites. The proposal is that each test item should contain:

- A sentence with an error in it
- The same sentence without the error
- The name of the error type
- Some indication of what forms of advice would be adequate to allow the correction process

It is desirable that a wide range of instantiations of the error type and its correction be generated, varying in grammatical construction, complexity and lexical choice. For grammar checking, a set of deliberately misleading inputs ("traps" for the system) could also be included.

Given the error taxonomy based on error sources quoted above, error examples can be generated from positive examples by means of the transformations specified in the error descriptions.

The effect of interacting phenomena is yet to be considered in any detail.

2.4 Neal Montgomery Method

The Neal Montgomery Method ([23]) is an evaluation procedure that is intended to be appropriate for any type of NLP system in any application domain and without modification for a particular test corpus or domain. The procedure helps the evaluator create, modify, or tailor test sentences for the particular NLP system, and score its performance.

It consists of a categorisation of linguistic features (lexical, syntactic, semantic or pragmatic) requiring processing capabilities, along with instructions for the formation of suitable domain-relevant sentences with these features and for scoring system output for this test material.

Each identified linguistic feature is defined and explained in order to establish a standard delimitation of that feature. Illustrative language patterns and sample sentences are provided to assist the human evaluator in formulating input which tests the feature on the NLP system within the system's native domain.

The methodology is designed to allow systems to be tested within their own domains, to avoid the cost of domain customisation for test purposes.

Classification scheme

The classification scheme consists of the categories listed below:

- Basic sentences
- Interrogative sentences
- Noun Phrases
- Adverbials
- Verbs and Verb Phrases
- Quantifiers
- Comparatives
- Connectives
- Embedded sentences
- Reference
- Ellipsis
- Semantics of Events

The classification scheme assumes a blackbox approach to evaluation, so a phenomenon such as anaphora is not subdivided under lexical/syntactic/semantic/discourse.

The classification scheme is tree-structured and includes verbal pointers to other sections of the classification scheme for classes that could be classified under multiple parents.

Test suite construction

The tool is designed to progress from very elementary sentence types containing simple constituents to more complex sentence (or paragraph) types. The idea is that each time a test sentence (or paragraph) is presented to the NLP system being evaluated, the sentence or paragraph should contain only one new (untested) linguistic capability or one new untested combination of tested capabilities.

Structure of test items

Each test item includes:

- an explanation and a definition of the linguistic phenomenon or capability being tested, along with any special instructions for testing.
- patterns/descriptions that define the structure and features of the test sentences to be composed and input to the NLP system under evaluation.
- example sentences to aid the evaluator in composing test sentences.
- a statement of the evaluation criteria for the individual test item.
- a place for the evaluator's test sentences.
- a place for the evaluator' score.

The examples in the tool are from two application domains: "employee database" and "terrorist domain".

The evaluator picks a sentence pattern from those listed in the particular test item, or from those in previous sections where the NLP system has succeeded. The evaluator then makes the substitutions described by the tool item: e.g. replace [NP] with ([Det])[Adj.Nom]. Then the evaluator replaces the non-terminals in the pattern with words that are within the lexicon of the NLP system being tested and appropriate to the domain of the system.

In addition to the procedure itself, evaluators who use the tool are encouraged to follow some guidelines including:

- submit multiple test inputs for each test item;
- use a variety in test inputs for each test item.[1]

2.5 The Sourcebook Project

The Sourcebook Project([28]) is part of the Artificial Intelligence Measurement System (AIMS) project of the Center for the Study of Evaluation at UCLA. The AIMS project aims to develop evaluation criteria for expert systems, vision systems and NL systems.

The NL part of the AIMS project has two parts:

[1] In cases where an evaluator used very similar inputs for three test items the score would be erroneously high or low with respect to the system's behaviour and with respect to the other evaluators' scores.)

1. to develop methods for describing the linguistic coverage of NL systems.
2. to develop methods for a "process evaluation" of NL systems.

   A process evaluation includes questions of efficiency, perspicuity and coverage of the range of concepts a system can handle.

The Sourcebook focuses on developing methods for describing the linguistic coverage of systems. It consists of a database of "exemplars" of representative problems in NL understanding, mostly from the computational linguistics literature. Each exemplar includes:

- a piece of text (sentence, dialogue fragment)
- a description of the linguistic/conceptual issue at stake
- discussion of the problems in understanding the text
- references to more extensive discussion in the literature.

The Sourcebook contains a conceptual taxonomy of the types of issues represented in the database, and the exemplars are indexed by their source in the literature and by the conceptual class of the issue so that the user can readily access the relevant examples. The Sourcebook thus provides a structured representation of the coverage that can be expected of a NL system.

The goals of the Project are stated as follows:

1. the criteria should be applicable over the broadest possible range of systems
2. the criteria should be independent of implementation issues including programming language
3. the criteria should be comprehensible to the general user and to researchers outside computational linguistics.

The exemplars are mostly taken from the literature on computational linguistics. However, it is hoped that by defining the issues through specific examples and conceptual classification, rather than implementation details or linguistic theories, the Sourcebook can be accessible to non-specialists in computational linguistics.

Classification scheme

There are three groups in the hierarchical classification which roughly match stages of development of NL systems. They correspond to (1) simple database query systems, (2) database systems capable of extended interaction, (3) systems where the knowledge flow between user and system goes both ways:

1. Single-utterance issues;
2. Connected-utterance issues including anaphora, ellipsis, reasoning, etc.

3. True-dialogue issues including recognizing user goals and plans, speech acts, etc.

The Sourcebook Project advocates a similar type of evaluation methodology to the Neal Montgomery Method in that it provides a collection of exemplars of numerous types of linguistic phenomena, but it does not provide a procedure or method for evaluating systems' performance on processing these phenomena.

The Neal Montgomery Method shares some similarities with The Sourcebook in that it provides:

- a hierarchically structured classification scheme for linguistic phenomena
- an emphasis on descriptions of the linguistic phenomena covered in the Tool
- examples illustrating linguistic phenomena

The Neal Montgomery Method has, however, a broader and more detailed coverage than the Sourcebook.

2.6 Heid and Hildenbrand

Heid and Hildenbrand([15]) describe an evaluation they have performed on the French to German module of the SYSTRAN MT system. They distinguish between two types of evaluation:

- General evaluation

  The aim of this type of evaluation is to test a system's performance with respect to a wide variety of input texts, according to a set of core phenomena which are not necessarily derived from the application profile.

- Application-specific evaluation

  This type of evaluation is interested in finding out how a specific system performs with a given type of input material.

The type of evaluation they performed on SYSTRAN was mainly "general" evaluation. The evaluation was not entirely blackbox, given that they had some informal information about the contents of the lexicon.

Test suites were constructed to test verb subcategorisation. Verbs were classified according to the classification scheme adopted by the developer. The idea was to test whether verbs that were similarly categorised by the developer displayed the same behaviour when handled by the system. An additional subclassification scheme that classified according to the types of modifications needed (e.g. the change of argument-complement relations, such as *plaire* $<->$ *like*) was used in part afterwards, in the analysis of results but turned out not to be relevant in all cases.

Two types of test suite were devised: a "basic" test suite and a "derived" test suite. The basic test suite contained a "trivial" sentence for each verb, which was in the present tense, and in the affirmative. The test suites had the following characteristics:

- The vocabulary used was very limited to avoid interference between lexical phenomena concerning the lexical material not being tested for.
- There were no categorial homographs (except *that* complementizer and demonstrative that were used only as a complementizer in the test suite).
- No negation was contained in the sentences.

The derived test suite was derived from the basic test suite in a modular way. Modularity meant that for each derived test suite, as few contextual parameters as possible were changed. So, for example, just the verb tense was changed or an adjunct added or the whole sentence from the basic test suite was embedded as a complement clause or a relative. No lexical changes of the minimal vocabulary used in the basic test suite were made. Each parameter change was individually applied to all sentences contained in the test suite to create several derived test suites equal in size to the basic one.

Besides modularization at the level of the parameters added to the basic test suites, another type of modularization was used - partial test suites for subsets of the basic test suites were generated, e.g. French reflexive and pronominal verbs, or verbs leading to German equivalents which take an obligatory expletive element. Thus characteristics of the target language were used in modularising the test suites.

Test suites containing combinations of parameters were then created to check the interaction of several phenomena.

2.7 Survey for *The Editor's Assistant*

Douglas ([5]) developed a set of tests to test a range of grammar and style checking programs. The purpose of the evaluation was two-fold: firstly, to assess the adequacy of the systems and to ascertain the commercial state of the art, but, further than that, to assess the kinds of technology used and how this related to the adequacy performances displayed. The second aim was to provide input to the design and development of *The Editor's Assistant*, a writers' aid.

Evaluation methodology

The two-fold aims of the adequacy evaluation are reflected in two linked taxonomies. The first taxonomy, the **System Centred Taxonomy**, aims to present a characterisation of possible underlying mechanisms used by different systems, on a number of dimensions:

1. representing the text,
2. recognising patterns in the text,
3. responding to errorful text,
4. manipulating the text representation, and
5. customising the rules.

Dimensions (1) and (2) were intended to reflect the overall potential capability of the system's underlying technology to detect and respond to grammatical errors; the division into how the text is apparently represented and what kind of patterns the error rules can find in the text was intended to clarify the level of linguistic sophistication of the underlying methods used by the various systems. These dimensions, and to some extent the third dimension (which reflects whether responses from the system seem to be all canned text or whether they have variable pattern elements) form a group of properly linguistic objects of evaluation.

The second taxonomy, the **User Centred Taxonomy**, provides a list of error types, intended to provide a system of categories by which the errors detected by systems would be classified. It is divided into four top-level sections, **Grammar**, **Punctuation**, **Style**, and **Usage**. Grammar errors are further subdivided into agreement and inflection errors, errors in the use of relative and reflexive pronouns, etc.

The two taxonomies are linked by the fact that both detecting and correcting particular errors (or examples of errors) were deemed to require a certain level of linguistic technology. Each entry in the grammar section of the User Centred Taxonomy contains a "requirements" specification, that is intended to reflect what underlying linguistic sophistication a system would need to successfully detect the error in question, and hence it provides a link from the user centred taxonomy to the system centred taxonomy. This link provides the basis for the composition of the test suites (see below), and their interpretation to give the results for the first two dimensions of evaluation.

The User Centred Taxonomy was thus used to position the system being evaluated within the System Centred Taxonomy, i.e., to determine, from the results of a set of tests based on the User Centred Taxonomy, what level of technology the system under test utilised, and thus its potential range of operation.

The test suite

The test suite was composed of an extension of the examples in the User Centred Taxonomy. Each error type has its corresponding set of examples in the test suite. Test examples were constructed in both positive and negative modes, that is, where an error exists in the construction and where one does not.

One implicit guide to the construction of these ranges of tests was to provide instances of the construction under consideration that might be thought to be exceptions to some over-simple general rule. One overgeneral rule, for example, is to treat all words that end in *-er* as comparatives, because *-er* is a standard comparative suffix. This rule is simple, because it can be operated simply with character-level regular expression rules, but it is often inaccurate (e.g. it would flag *more danger* as a comparative error on the model of *more better*). Similar tests were included for both false positive and false negative responses wherever they could be devised and were thought to be useful.

Two strategies for detecting the effects of such over-simple rules could be envisaged, either rather exhaustive testing with a large amount of lexical variation, or guessing which rules are likely to be used given the technology available, and devising tests based on knowledge of their limitations. Mostly the latter route was taken.

An error taxonomy for Progress evaluation

The system-based aspect of the taxonomy of errors that was developed for adequacy evaluation was the basis for the progress evaluation scheme for *The Editor's Assistant.* The analysis of the kinds of error found led to the development of a group of grammar error types, viewed from a computational linguistics point of view. These error types are as follows:

Constraint Violation Errors: These involve what, in most contemporary syntactic theories, are best viewed as the violation of constraints on feature values. All agreement errors fall into this category.

Lexical Confusion: These involve the confusion of one lexical item with another. For example: confusion of *its* and *it's*; confusion of *there*, *their*, and *they're*; confusion of possessive *'s* and plural *s*.

Syntactic Awkwardness: These are cases that are not ungrammatical, but are stylistically awkward or likely to cause processing problems for the reader. (e.g. multiple embedding constructions potentially ambiguous syntactic structures, and garden path sentences).

Missing or Extra Elements: These are cases where elements (either words or punctuation symbols) are omitted or mistakenly included in a text. For example: Unpaired delimiters; Missing delimiters; Missing list separators; Double syntactic function.

As with the user-centred taxonomy, a link was made between the system internal mechanisms. The error types are associated with general error detection/correction mechanisms in *The Editor's Assistant.* The third type (associated with syntactic awkwardness) was similar to pattern-recognition algorithms, since it merely recognised patterns in the analysed text. The other three types were based on different error hypothesis schemes, and relied on an extra processing loop to test whether performing the suggested correction on the hypothesized error actually led to a better sentence.

2.8 Aerospatiale methodology

Aerospatiale describe a methodology for test suite construction ([29]) that is divided into three stages, and which aims at the production of a "generic" test suite.

Stage 1

Aerospatiale derive their test suites from an error classification that is obtained from comparing system output with proofed text. The test sentences are extracted from the source text, and then they are aligned with their proofed equivalents.

The error classification adopted is:

- lexical/terminological
- morphological
- syntactic

- segmentation

Given the blackbox scenario which is assumed in this setup, a more practical classification of errors with regard to the system's modules (dictionaries, analysis, transfer, generation, format module, etc.) is not possible.

The advantages of this test suite design are:

- the fact that the errors are known to occur, and are therefore not artificial in any sense;
- the test suite gives a precise idea of the lexical and terminological adequacy of the system.

The disadvantage is that the test suites generated are specific to the text type of the input text.

Stage 2

There are two steps used to create generic test suites:

- Simplify and reduce the vocabulary used.
- Isolate the phenomena encountered in the processed sentences to obtain a general linguistic representation of them.

Individual phenomena in the input sentences are isolated and for each phenomenon a sentence as short and simple as possible is built to exemplify it, maintaining where possible, the original vocabulary (particularly if the phenomenon is likely to be vocabulary dependent) or the original layout.

Stage 3

For each phenomenon, a set of test sentences is written that are intended to illustrate all possible linguistic variations of the phenomenon (in terms of lexical, morphological, syntactic, semantic and pragmatic variations).

There are two main concerns to the industrial evaluator:

- how to classify the phenomena
- how to be exhaustive

Classification

- By Parts of Speech
- By error taxonomy
  This can be by the origin of errors detected:
  - analysis

  - transfer
  - generation
  - lexical transfer

  However, this assumes a glassbox evaluation scenario or at least assumes that the developer agrees to cooperate and exchange information with the industrial evaluator.

- By the type of intervention necessary to perform corrections.

2.9 DITO

The DITO test collection (see [24]) is included in this review not because it is elaborated within a certain evaluation framework (in fact, the authors are careful to point out the differences between diagnostic evaluation for which the test suite is designed and system evaluation in general), but because it represents an interesting approach to the construction of test suites.

The DITO test suite is designed as a diagnostic evaluation tool for the recognition and characterisation of problems of syntactic analysis. The material consists of artificially constructed sentences that illustrate syntactic phenomena. A characteristic of the suite is that test items are:

- systematically constructed,
- annotated with information about the syntactic phenomena illustrated.
- mounted on a database for ease of manipulation.

Several factors were borne in mind while constructing the syntactic examples.

- limited size of the lexicon.
- adherence to the following standards: (somewhat) formal, conversational High German, i.e. colloquialisms, literary peculiarities, and regional dialects are avoided.
- selected testing of negative examples. The test suite was kept small, but not at the cost of using great ingenuity to create minimal sets of testing data, nor at the cost of introducing very unnatural examples into the test catalogue.
- minimization of irrelevant ambiguity (bearing in mind that it cannot be fully eliminated).
- attention to analytical problems. Not only the constructions, but also the problems known to be difficult in their analysis were catalogued.

The primary goal is that the coverage of the phenomena be as comprehensive and as systematic as possible.

One of the most original aspects of the DITO test suite is its detailed annotation scheme. Not only are linguistic phenomena annotated for, but the annotation scheme also contains information about the precise structure of the sentence such as the position of the finite verb and the position of other phrases. The aim was to focus on properties that are relatively easy to test and about which there is little theoretical dispute, but that nevertheless provide a reasonable reflection of a system's accuracy.

The advantages of developing the database are quoted as follows (see [19]):

- to provide a concise organisation of syntactic data
- to ease access to syntactic information
- to maintain consistency
- to allow variable logical views of the data
- to facilitate an efficient extension of the syntactic material to treat further areas

2.10 Gambäck

Gambäck ([11]) describes two evaluation methods, one called *compositionality evaluation*, which is aimed at the diagnostic evaluation of rule-based systems, and the other called *phantom evaluation*, which is a more general system-testing strategy. The first evaluation method makes use of test suites, the second one of test corpora.

Compositional Evaluation

The compositional evaluation method was devised for evaluating the compositionality of a transfer-based machine translation system, and demonstrated on the BCI (Bilingual Conversation Interpreter) [1], an interactive transfer-based bidirectional system developed in a co-operation between SICS and SRI Cambridge. The main components of the BCI are English and Swedish versions of the SRI Core Language Engine ([2]), transfer taking place at the level of Quasi Logical Form (QLF); the transfer formalism is unification-based and bidirectional.

For compositionality to be a meaningful notion in the first place, it must be possible for rules to apply to partial structures. These structures can consequently occur in different contexts; other rules will apply to the contexts as such. The aim of the evaluation was to find out which particular combinations of rules and contexts gave rise to special problems. *Compositionality tables* (see 2) were constructed, in which a set of rules and a set of contexts were systematically combined in all possible meaningful combinations. This was done in order to discover the extent to which the complex rules continued to function in the different contexts. An example of these tables is given below: Table 3 gives a set of rules, which exemplify six common types of complex transfer. Table 4 gives a set of twelve common types of context in which the constructions referred to by the rules can occur.

Full-scale compositionality testing

Gambãck points out that the compositionality table (Table 2) is too small to detect more than a fraction of the bad rule interactions, and he stresses the importance of extending the set of contexts in a systematic fashion. He suggests that a full set of contexts should include at a minimum all legal combinations of independent choices along the following dimensions:

Table 3: Compositionality Table (Swedish → English shown above English → Swedish)						
Transfer context	Different particles	Active to passive	Verb to adjective	Support verb to normal verb	Single verb to phrase	Idiomatic use of PP
Present tense	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
Perfect tense	OK OK	generator OK	OK OK	OK OK	OK OK	OK OK
Negated	pres-not pres-not	pres-not pres-not	pres-not pres-not	past-not past-not	pres-not pres-not	transfer transfer
YN-question	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
WH-question	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
Passive	OK OK	- -	- OK	- -	- OK	- -
Relative clause	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
Sentential complement	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
Embedded question	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
VP modifier	OK OK	transfer transfer	OK OK	OK OK	OK OK	OK OK
Change of aspect	OK OK	OK OK	OK OK	OK OK	OK OK	OK OK
Object raising	transfer OK	transfer OK	transfer OK	transfer OK	transfer OK	transfer OK

Table 2: Table 3 from Gambãck

Complex Transfer Type	English Swedish example	
Different particles	John likes Mary	John tycker om Mary
Passive to active	Insurance is included	Försäkring ingår
Verb to adjective	John owes Mary $20	John bf är skyldig Mary $20
Support verb to normal verb	John had an accident	John råkade ut för en olycka
Single verb to phrase	John wants a car	John vill ha en bil (lit: wants to have)
Idiomatic use of PP	John is in a hurry	John bf har bråttom (lit. has hurry)

Table 3: Examples of Complex Transfer types

Transfer Context	English Swedish example	
Present tense	John likes Mary	John tycker om Mary
Perfect tense	John has liked Mary	John hat tyckt om Mary
Negated	John doesn't like Mary	John tycker inte om Mary
YN-question	Does John like Mary?	Tycker John om Mary?
WH-question	Who does John like?	Vem tycker John om?
Passive	Mary was liked by John	Mary blev omtyckt av John
Relative clause	The woman that John likes	Kvinnan som John tycker om
Sentential complements	I know that John likes Mary	Jag vet att John tycker om Mary
Embedded question	I know who John likes	Jag vet vem John tycker om
VP modifier	John likes Mary today	John tycker om Mary idag
Object raising	I want John to like Mary	Jag vill att John ska tycka om Mary (lit: I want that John shall like Mary)
Change of aspect	John stopped liking Mary	John slutade tycka om Mary (lit: John stopped like-INF Mary)

Table 4: Examples of Transfer Contexts

- *Tense*: Present, past or future.
- *Mood*: Active or passive.
- *Negation*: Positive or negative.
- *Modification*: Unmodified, PP modification, ADVP modification, modified by fronted constituent.
- *Clause-type*: Declarative sentence, Y-N question, WH-question, relative clause, sentential complement, embedded question, progressive VP complement, object raising.

Gambăck calculates that multiplying out all the choices gives a total of 384 distinct contexts; this must then be multiplied by the number of transfer rule types to be tested, and doubled to get both directions of transfer. With the figures given above, 4608 sentences would have to be tested. In practice, of course, not all combinations are possible (eg. passives do not interact well with other rule-contexts). When impossible combinations are eliminated, there is still a total of 2809 sentences.

2.11 Lehrberger and Bourbeau

Lehrberger and Bourbeau [21] describe a methodology for MT evaluation from the point of view of the user that mainly concentrates on linguistic evaluation. They propose two steps in this evaluation process: the first consists of submitting test sentences to the system, the second consists in choosing sample texts for submission to the system.

Unlike the methodologies described above, Lehrberger and Bourbeau propose that the test suite should consist of sentences derived from real corpora, and not be artificially derived. The sentences should contain a representative set of sentences from a given domain. They have, however, no concrete suggestions about how to choose this set of sentences beyond studying the linguistic characteristics of the texts from which they are to be taken. Likewise, the choice of test corpora is based mainly on “intuitive criteria”.

Lehrberger and Bourbeau do however have more to say about the interpretation of results. Interpretation of results requires the following:

- a copy of the original corpus that was transcribed onto the computer
- a copy of the machine translation
- a copy of all the entries in the dictionary or dictionaries used by the system to translate the words in the corpus
- if required, traces of certain sentences translated by the system.
- a copy of the human revision of the machine translation
- a copy of the translation of the original corpus made by a human translator.

All translation errors are noted and classified according to linguistic phenomenon, where a distinction is made between phenomena specific to the comprehension of the source language and those specific to the transfer of this comprehension into the target language.

Lehrberger and Bourbeau claim that, even if the evaluators have no access to the system's grammars, they may still determine the following information, provided that they do have access to the system dictionaries and to processing traces for the sentences:

- whether the system makes a local or complete analysis of the sentence
- whether the system analyses both the ‘surface’ and ’deep’ structures of the sentence
- whether the system translates by direct transfer from the source language to the target language
- whether the system translates by transferring from the source structure to the target structure by means of a pivot structure
- how the system interprets the linguistic information from the dictionary or dictionaries
- the principal linguistic operations of each component
- the degree of comprehension of the various linguistic relations between each of the sentence constituents
- linguistic phenomena not processed by the grammars of the system
- the linguistic phenomena for which processing or the proposed description is incomplete or inadequate
- the boundaries between analysis, transfer and generation.

2.12 Discussion

Certain facts about test suites emerge from the above discussion:

- There is a link between the functionality of a system and test suite design.

  Both the Gallier Sparck Jones and EAGLES ([10, 6]) evaluation frameworks start from the notion that it is important to understand the functionality of a system before you can develop ways to assess it. Quality characteristics (EAGLES) and criteria (GS) can be decomposed into "attributes" (EAGLES) and "measures" (GS). These last two categories describe those things whose value is of interest to the user, and values for them are arrived at either by inspection or by testing. Test suites are an example of test material to test the linguistic functionality of a system.

- Test suite construction requires an analysis of systems according to not only their function, but also (following GS terminology) to their parameters and variables.

  In other words, test suite design may be sensitive not only to system architecture (parameters) but also to system input (variables). Test suites can of course be constructed that make no reference to the internal workings of a system (wide-bound evaluation test suites, in GS terminology). Note also that some test suites may choose to concentrate on system variables rather than system parameters or vice versa. For example, the SAM ([10]) project test data is designed to cover a whole range of speech-relevant factors. It does not follow that the SAM environment factors are necessarily relevant to specific applications.

Hence much of the variation in coverage of test data (both depth and breadth of coverage) in the test data described above is attributable to different evaluation types (e.g. glassbox versus blackbox evaluation), system type and hence system function, and system architecture.

In Work Package 1 of TSNLP (WP1), we identified systematicity as an important aspect of test suite design. Systematicity is a particular focus of the test suites of [15, 23, 11, 24].

The test suites described above differ not only with respect to content but also with respect to presentation. Different classification schemes, for example, are appropriate for different evaluation scenarios: Some of those described above include:

1. Input-related source of errors
   (e.g. category of author, type of modality of input text)
2. System-related origin of error
   (e.g. analysis/transfer/synthesis)
3. Part of speech
4. Type of intervention needed to perform correction

Note that (2) is only possible in a glassbox scenario.

The above reviews substantiate one of the findings of WP1, namely that very few test suites concern themselves with documentation. Exceptions are [23, 28]. Likewise, very little attention is paid to annotation schemes (again, with the exception of [24]), although this is very important for reusability.

The overview of test suite design and construction given above substantiates another finding of WP1, namely the lack of tools to support test suite use and construction. Only [23] and [24] have anything to say about tools.

Finally, the important aspect of interpretation of test suite results is only touched upon and not discussed in any great depth (except [21]) although, as discussed in WP1, interpretation of results is vital to assessing test suites as an evaluation tool.

3 Evaluation type

The EAGLES Working Group on NLP Evaluation and Assessment have identified three different types of evaluation and their respective aims, which we outlined in the WP1 report, but which we repeat here for convenience:

- *diagnostic evaluation*, which aims at localizing deficiencies;
- *progress evaluation*, for a comparison between successive stages of development of a system;
- and *adequacy evaluation*, to determine whether and to what extent a particular system meets some pre-specified requirements.

These categories of evaluation types are not independent but related. In some ways the relationship between them can be seen as that of inclusion. Adequacy evaluation attempts to find out whether a system is suitable for a specific application. Therefore it has to know where the deficiencies of the system lie and what its potential in terms of development over time is. Progress evaluation requires diagnostic evaluation and since no NLP system is perfect right from the start, progress evaluation should be an essential part of adequacy evaluation.

Progress evaluation makes sense only if you know in what aspects progress has been made. For that purpose you have to know what deficiencies there were and which of them have been removed over time. Identifying deficiencies is a matter of diagnostic evaluation.

Thus adequacy evaluation must include diagnostic evaluation and progress evaluation, and progress evaluation must include diagnostic evaluation.

To the extent that developers are developing products which can be sold on the market, then adequacy evaluation will eventually become as important for a developer as for a user, even if diagnostic evaluation may play a more important role in the earlier stages of development.

Nevertheless, it is perfectly possible to write test suites for a combination of different evaluation types. We can identify the following different parameters and levels where the evaluation type (either diagnostic or adequacy) has a direct impact on the process of designing test suites:

- The conditions of testing:
  - Blackbox vs. glassbox situation:
  - Availability and definition of pre-specified requirements
  - Definition and measurement criteria of an acceptance
  - Types of texts involved

  - Languages and/or language pairs involved

- The characteristics of the test items:
  - The origin of the test items
  - The nature of the test items
  - The coverage of the test items
  - The combination vs. isolation of the test items

- The testing procedure:
  - A unique vs. several sets of test sentences

3.1 The conditions of testing

The conditions and the environment of testing or evaluating, especially the five parameters below, may have a great influence on the process of building test suites:

- Blackbox vs. glassbox situation:

  Depending on whether the test suite designer has an extensive knowledge of the whole system under evaluation or has only access to the interfaces, they will adopt a different strategy of testing and will focus on different test items. Most of the time, in a glassbox situation, the diagnostic evaluation strategy will be chosen by testers, be they developers or only involved in the specification phase of a system under development. Testers confined in a blackbox situation will mainly blindly evaluate blindly existing systems and in order to compensate for the lack of information on the system, they will try to set their own evaluation scale and references as well as acceptance level, using their particular textual and working environment. They will thus perform an adequacy evaluation.

- Availability and definition of pre-specified requirements:

  Before actually designing test suites, it may be necessary for the test suite designer to define the basic objectives and requirements of the evaluation. In the case of an adequacy evaluation, the most straightforward pre-requisite is that the system under evaluation should be able to handle correctly the real texts to be processed and it should also be easily integrable in the overall existing environment. It should be noted that adequacy evaluations are most of the time carried out within, or on bealf of, industrial companies. In the case of a diagnostic evaluation, the initial requirements which may or not be defined are more concerned with the system itself, i.e. the identification of its current shortcomings and possibly some indications on the amount and chances of success of any further developments or improvements.

- Definition and measurement criteria of an acceptance level:

  This parameter is supposed to deal with the following questions: What are the expected results of the evaluation and what is the tolerance threshold in case of inadequate results? For a TS designer involved in an adequacy evaluation, the answer to the first question is guided by the contents of the texts he wants to process with the system under evaluation. If he has a human reference (e.g. human translations), the objective is to make a quantitative and qualitative comparison between the human results and the results produced by the NLP system under evaluation. The tolerance threshold will be defined according to the actual use and frequency of test items in the real texts (through statistical weightings). In the case of a diagnostic evaluation, the definition of an acceptance level may not be crucial.

- Types of texts involved

  The notion of texts is only relevant in cases where the tester actually wants to use the system environment in an operational context. It mainly concerns the adequacy evaluation. In such a case, we can identify different scenarios of selection of texts to be processed by the system under evaluation:

  – No selection at all
    – > the system should be able to process any text: the TS designer will try to be as exhaustive and general as possible while choosing the test items.
  – Selection on activity or terminological criteria
    – > the system should be able to process any type of texts dealing with a particular subject field: the TS designer will focus on the testing of lexical and terminological linguistic variations and technical relevance.
  – Selection on writing or editorial criteria
    – > the system should be able to process a particular type of texts (e.g. maintenance manuals, press articles, contracts) dealing with different subject fields: the TS designer will focus on the testing of syntactic (repeated) structures.
  – Selection of texts on the basis of a combination of the above criteria.
  – Languages and/or language pairs involved:
    A distinction should be made between monolingual test suites designed for a multilingual application (e.g. MT system) and bi- or multilingual test suites. In the first case, the design of the test suites in the source language may be influenced by the expected results in the target language. In the case of multilingual test suites (of which we could not find any in our preliminary collection of existing TS in WP 1), the TS designer has an additional constraint: he has to build or extract test items across languages. He may identify scores of multilingual phenomena which remain stable across languages or he may adapt the test suites to the specificity of each language.

3.2 The characteristics of the test items

- The origin of the test items:

  For a diagnostic evaluation, test items and test sentences are most of the time artificially constructed or extracted from general, application-independent texts. For an adequacy evaluation, test items and test sentences are either directly extracted from real text (most of the time technical documents) or have been constructed on the basis of real texts.

- The nature of the test items:

  In the case of a diagnostic evaluation, the linguistic phenomena involved in the test suites are representative of the general language, and the TS designer will try to be as exhaustive as possible in order to illustrate all the linguistic phenomena which may occur in any situation of communication (written/oral, colloquial/standard, etc.). Extra linguistic phenomena may be included where they are sufficiently relevant and general. In the case of an adequacy evaluation, the TS designer concentrates on a subset of phenomena identified in real texts. The test items also include extra-grammatical input which may be specific to a particular sub-language. As they are representative of technical texts, they may also include extra-linguistic phenomena (such as punctuation, lists, acronyms, etc.).

- The coverage of the test items:

  Testing the interaction of phenomena can be important in both diagnostic and adequacy evaluation. In adequacy evaluation, if the test sentences are directly extracted from real texts, the test sentences are likely to contain several linguistic phenomena.

3.3 The testing procedure

There can be either a unique or several sets of test sentences.

For a diagnostic evaluation, only one set of test sentences will be designed for a wide spectrum of generic linguistic phenomena. For an adequacy evaluation, it may be useful and necessary to build and apply a second set of test sentences on the basis of the results obtained with the first test suite. The second test suite will then concentrate on test items which have produced inadequate results; the goal of the second set is to examine as deeply as possible the behaviour of the system under evaluation with respect to a subset of specific phenomena and, as such, this test suite is supposed to cover, for each phenomenon, all the possible variation cases (so as to avoid, or at least reduce, the random aspects).

4 System/application/component type

A variety of different types of test suites can be identified depending on the basis of what they are designed to test:

1. A specific system (e.g. the Systran MT system)
2. An application-type (e.g. MT systems)
3. A component (e.g. the synthesis module of a MT system)
4. A range of applications (e.g. parsers, grammar checkers)

4.1 System-level test suites

As [10] notes, it is important to differentiate between generic and non-generic systems. Generic systems are designed for some task type and while they may incorporate a domain-independent knowledge base, they do not include domain-specific resources. Components of systems may also be generic. For example a generic language processing component may include a general-purpose syntactic grammar of some language and a core lexicon of common words, but not domain-specific vocabulary.

Generic systems pose a particular problem for evaluation since it is impossible to evaluate any system fully unless all the resources required for its application are present. An example of a generic system is the Core Language Engine (CLE) ([2]), which consists of a syntactic and semantic parser that delivers logical form representations of meaning. While generic systems may be difficult to evaluate it is nevertheless important to be able to evaluate them to assess, for example, their potential usefulness for some application. Thus customisation is an important aspect in their evaluation. The grammar and lexicon of a generic system will be "core" in some sense, and a test suite designed to test it will reflect this.

Specific system level test suites on the other hand will typically concentrate on testing whatever the intended or desired coverage of the system to be tested is. Phenomena beyond this coverage may be included selectively to test how the system will behave in their presence.

System level test suites will typically aim at in-depth testing of all phenomena covered, both with positive and negative inputs. For reasons of systematicity and efficiency, test items need to be ordered in progressive sequence from simple to complex, and we may also insist on the principle of introducing only one new phenomenon per test item.

4.2 Application level test suites

This is a very common type of test suite, that will be used in for example an adequacy evaluation of a range of systems belonging to the same application type. The lexicon and phenomena contained in the test suite can be geared towards the task the application is designed to handle. Thus MT system test suites will typically contain problems that are known to be hard for translation, including cases of lexical and structural mismatches, ambiguity, etc. Lexical confusability based on spelling and grammatical categories (e.g. *there*,*their*, *they're*) will feature in test suites designed for spelling checkers and grammar checkers, while lexical confusability based on the sound of words will feature in test suites for speech recognition systems.

Application level test suites will typically be organized in progressive order, but the depth of testing may vary over phenomena, depending on estimated degree of centrality of a phenomenon or on its known linguistic intricacy.

4.3 Component-level test suites

Flank et al. ([8]) describe some problems associated with module-level testing. They recommend that extreme care be taken in choosing the test suites so they test capabilities in a targeted way. For example, discourse tests should be embedded in sentences that can be parsed by the system, and vocabulary should be restricted. They conclude that module-level testing is possible only when the system is designed in a modular way and when the interfaces between modules are clearly defined.

Note that there is frequently a problem as to how to classify phenomena, whether as syntactic/semantic (e.g. anaphora) or as grammatical/lexical (e.g. the distinction between grammar and spelling errors is sometimes hazy). Where a particular phenomenon is handled is therefore frequently system dependent.

Galliers and Sparck Jones ([10]) note that the problem with component testing, just like with generic system testing, is that it is incomplete, in the sense that for them, evaluation starts with assessing the system task. In particular, they emphasise the danger of extrapolating from component performance to system performance.

Nerbonne ([25]) reports that, in order to evaluate system components, it is necessary to identify their respective inputs and to charaterise their actual and desired correct outputs in some theory neutral and clear way (this is clearly important at least in comparative evaluation). He emphasizes the conditional nature of the results of component evaluation: only when the inputs from the prior syntactic component to the semantic module are known to be correct, can the semantic component's own contribution be considered.

To date, the most clearly defined methodology has been defined for the evaluation of syntactic processing components (parsers). It involves comparing system output with a standard parse from the Penntreebank (see eg. [13] and [14]). The problem for semantic evaluation is the lack of concensus over semantic representations.

4.4 General purpose test suites

General purpose test suites present a range of problems. Gallier and Sparck Jones ask whether we can find evaluation criteria and strategies that are independent of task and setup types, given the following two problems:

- that of distinguishing NLP from a larger task as far as the system itself is concerned, and
- that of distinguishing the system from its setup, especially in relation to the activities of its human users,

Flickinger et al. ([9]) amongst others, believe that it is impossible to devise a test suite to test the whole gamut of NLP systems. Nevertheless they have devised a test suite that is intended to test a wide range of NLP systems. It is characterised by a restricted vocabulary of some 200 items restricted to the field of office management.

A key issue with general purpose test suites is their flexibility and their ease of customisation. The Neal Montgomery Method ([23], see 2.4) addresses these issues. It is precisely designed to allow systems to be tested within their own domains, to avoid the cost of domain customisation for test purposes. The issue of customisation is explored further in WP2.1 deliverable "Test Suite Design and Methodology".

4.5 Discussion

As the above review illustrates, there are a number of dimensions along which test suites differ, according to what it is they are designed to test, including:

- coverage of phenomena, and
- depth of testing of each phenomenon.

It is also clear that the interpretation of the output differs with system/application/component type. (Automatic) comparison of expected and actual results can most readily be imagined for system level test suites. It is more difficult to generalize over the possibilities for application-level or general purpose test suites, since it will largely depend on the application area (including type of system output) or on the degree of user specificity. It is thus imaginable to automate comparison for a specific test suite for MT systems in as far as the required translations can be specified, whereas it seems harder to pre-define automatically comparable expected outputs for a class of systems like e.g. present day grammar checkers.

This has consequences for the annotation schemes of test suites, of course. It is also obvious that different types of test suite require different types of annotation, particularly with regard to the expected output.

5 System input

The relationship between system input and test suite design is a very important one, since it affects the coverage of testing and the depth of testing. There are several dimensions along which one can classify input text:

- Text type
- Authorship
- Modality

5.1 Text type

We can define text types according to:

- subject domain (e.g. business)
- function (e.g. letter)

Thus one text type might be "business letters", another "airline manuals". Each text type has its own particular vocabulary and conventions, ranging from syntactic constructions through to formatting conventions.

5.2 Authorship

This is an important distinction in some application areas, e.g. grammar checking, where the type of error that the system is meant to check will depend to a large extent on the competence of the authors, which may be affected by such things as whether or not they are native speakers of the language.

5.3 Modality

Experimentation with different modalities was conducted at the tutorial on "Evaluation of NL modalities" at the Third Conference on Applied Natural Language Processing in Trento, 1992 ([10], pp117-118). Modality differences which were then discussed included:

- Speech versus non-speech

  Speech is faster, more verbose, more tightly interactive, less planned, more repetitive and dysfluent; it uses less vocabulary, uses more pronouns, and is less syntactically complex and well integrated and more indirect than non-speech.

- Handwriting versus type-writing

  Handwriting is faster, less verbose, has fewer pronouns and verbs, and has longer turns with fewer interruptions than using a keyboard.

- Human-human versus human-computer communication

  Compared to human-human communication, human-computer communication has clearer pronunciation, slower delivery, briefer utterances, is more precisely specified, uses simplified grammar and vocabulary, is cautious and conservative in style, involves less presumption of common ground, comprises more limited and stereotyped dialogue and lacks confirmatory and explanatory language.

Kukich([20]) similarly notes that modality differences occur depending on whether the text is originally typed by a human, is to be read out by a text-to-speech synthesis application (e.g. Telecommunications Devices for the Deaf (TDD) applications), or is based on optical character recognition (OCR). For human-generated errors, research has brought to light error patterns that vary from the typical typos to typical misspellings, phonetically-based errors, run-on or split words, letter swapping, the influence of keyboard layout, etc.

Note that for speech understanding further characteristics of input speech are relevant:

- Speaker numbers and categories (including sex)
- Recording conditions
- Speech style (running on, with pauses, etc)

5.4 Discussion

As mentioned above in section 2.2, what is important is that a test suite be valid and appropriate, i.e. that it represents what it is supposed to represent and that it is appropriate as test data for the system/component under consideration. Thus a test suite must reflect the different text dimensions discussed above if it is to be representative of the system input. Of particular importance is to know the frequency of occurrence of phenomena and vocabulary in order to assign weightings. Some tools exist for frequency analysis of corpora, but exact characterisation of text remains a difficult problem.

General purpose test suites on the other hand are intended to be used in a variety of domains and possibly for a variety of systems and/or application types. Thus their vocabulary should be as general as possible. The Flickinger ([9]) test suite, for example, restricts its vocabulary to the general domain of office management. Construction types and weightings should reflect "general language", as much as this can be defined.

6 System architecture and test suite design

As mentioned above in section 2.2 it is possible to ignore intentionally system architecture when devising test suites. [10] describe "wide bound" evaluation, where performance criteria are adopted which make no reference to internal system design. By means of illustration they invite one to imagine an approach to the transcription of read text which identifies word boundaries by acoustic indications and so does not require a lexicon. They contrast that with a "narrow bound" evaluation for a lexicon-based system, where reference can be made to a design goal for the system which sets a target percent of words to be captured given an assumed vocabulary coverage in the lexicon: i.e. for a text system it might be expected that some words would never have been encountered before.

The notion of system-architecture related evaluation is closely related to the distinction between blackbox and glassbox evaluations (see [27]). Glassbox evaluation attempts to look inside the system and find ways of measuring how well it does something. It should measure the system's coverage of a particular linguistic phenomenon or set of phenomena and the data structures used to represent them. A blackbox evaluation, while having no access to the internal workings of a system, may also be interested in its architecture, from the point of view of assessing the potential of a system. In a blackbox scenario, however, the architecture has to be guessed at, since it is not known. The Editor's Assistant Project (see section 2.7 above) is an example of such a blackbox scenario, as is Lehrberger and Bourbeau's method (see section 2.11).

An example of a glassbox evaluation scenario where the test suites were written very much with the system architecture in mind (namely the grammar rules) is described by Gambäck (see section 2.10 above). Below we present Hoard's ([16]) ideas on how to categorise NLP systems with a view to constructing glassbox test suites and metrics for ranking a system's semantic and pragmatic capabilities. The approach is suitable, he says, for both progress and adequacy evaluation:

- **The intended meaning of the semantic representation.**

  Contrast a semantic representation such as *loves(John,Mary)*, where the interpretation of the roles is implicit, with a representation such as *loves(cognizer:John, range:Mary)*, where the interpretation of the roles is explicit and fixed. In the latter case, tests could be devised that assess the "self-awareness" of the system in the sense that we might suppose that such a system could answer questions about *how it knows what it knows*.

- **The closed- or open-world assumptions that are made and the values that the logic is capable of supporting.**

  For instance, in a system that has a Prolog-style logic, "yes" means "demonstrable from the information at hand", while "no" means "not demonstrably yes". A system with a three-valued (or higher) logic will support an open-world assumption. Thus, given information that *All men are mortal*, *Socrates is a man*, and *Jill is not a man*, a two-valued system will respond to the query *Is Jill mortal?* with "no", but it will also report "no" to the query *Is Sam mortal?*, about which we have, by hypothesis, no information. A three-valued system ought to respond with "don't know", or the equivalent, to the last inquiry. Systems that support fuzzy or probabilistic logics might or might not make open-world assumptions. If they do, it is fair to test them over such a suite of sentences.

- **What kind of valid conclusions can be drawn using the logic and reasoning system.**

  Rather than care about whether a system uses deductive, inductive, or abductive inferencing strategies, an evaluation should worry about the sorts of conclusions that are supported. Inferencing axioms include "X means Y" (e.g. *kill* means *cause to die*, "X doesn't mean Y" (e.g. *kill* doesn't mean *injure*, "X includes Y" (e.g. *kill* includes *murder*, "X is similar to Y" (e.g. *convince* is similar to *persuade*).

- **What finitary assumptions are made.**

  The stability of a system under different test situations may cause us to exclude certain kinds of sentences or dialogues if queries are not guaranteed to terminate in finite time or with available storage.

- **What semantic primitives are presumed.**

  If a system contains notions such as "locative, temporal, purposive, agent, goal" etc., then distinctions could be tested for such as the differing roles that *Bill* and *algebra* play in *John taught Bill* and *John taught algebra*, as well as *John taught Bill algebra*.

- **Whether the semantic representations are intended to be partial or complete descriptions of sentences (propositions, discourses, etc.) and whether the representations support both full and elliptical constructions.**

  How do systems represent *John is reading* where it is implicit that John is reading something?

- **What kind of well-formedness conditions apply to the semantic representations.**

  Systems should be tested to see if they exclude impossible semantic structures.

- **Whether an arbitrary or non-arbitrary mapping is assumed between syntactic and semantic representations.**

  Tested here is the system's granularity and completeness of representations.

- **The kind(s) of knowledge on which pragmatic interpretations are based.**

  The kinds of, and extent to which, knowledge bases (e.g. lexical, general knowledge) available for reasoning will effect the type of test suites constructed. It is pointless to evaluate system performance over metaphors and idioms if no system has any general way of dealing with them.

- **The types of inferencing that are supported.**

  Of especial interest here is whether a system can provide relevant answers to a query. For example, if a system "knows" that John convinced Bill of something or other, is it capable of noting the relevance of the query *Did John persuade Bill*.

- **Whether reasoning about classes (i.e. higher order rules) is available.**

  Can the system recognize and exploit equivalent and related phenomena by virtue of having mechanisms to relate semantic representations? For example, could a system recognize that *Did John close the door?* is true if it knows that *It was John who closed the door* is true, i.e. can it recognize the equivalence of cleft and unclefted sentences?

- **Whether the full range of natural language quantification is handled.**

6.1 Statistical Based Systems

So far we have been talking about systems whose architecture was based on linguistic rules. However, there exist statistical based systems which are based on a different source of knowledge. Linguistic rule based systems use linguistic rules that try to formalize the rules of grammar, whereas statistical based systems make decisions concerning the structure of a sentence on the basis of statistical information that is extracted from large corpora.

The test suite is generally accepted as a useful tool for diagnostic evaluation of linguistic rule based systems under development (see for example [24]). It is a collection of test data that has been drawn up with the intention to systematically test linguistic phenomena, and which allows a system developer to establish which of the linguistic rules in the system do not work adequately. It is also well-known that it can be useful for other types of system assessment (especially adequacy evaluation) as soon as the evaluator has access to frequency information from typical input texts.

As for statistical based systems, it is not at all clear whether a test suite could be useful in evaluating them. On the basis of the fact that one does not write linguistic rules in statistical based NLP system to cope with natural language phenomena, one could decide that in principle the test suite has got no place whatsoever in statistical based NLP. Still, this type of system does not do away completely with the notions introduced by grammars; sometimes it might just deal with them in a different way. Therefore, it might be interesting to establish to what extent test suites could provide useful information with respect to the performance of the statistical predictions related to those structures.

6.1.1 Current Techniques in Testing Statistical Based Systems

Publications on the evaluation of statistical systems generally report that they use test corpora to assess system performance. The most interesting type of test corpora seem to be those which are tagged or bracketed. Tagged corpora have already been used for system (statistic and non-statistic) evaluation, for example: [4] uses a manually bracketed corpus (Lancaster Treebank) for the development and evaluation of a probabilistic grammar. At the University of Sussex, Geoffrey Sampson (personal communication) used a subset of the (annotated) SUSANNE corpus to test his APRIL parser. For SUSANNE, tools have been developed to locate configurations of interest in the tree structures of the analysed corpus. Slightly different is the technique used by Wojcik et al. [30], who report on the use of a 'spe' program in evaluation, which takes natural language sentences and produces a bracketed version of them. Both the bracketed and unbracketed version are used in evaluation.

The reason for using test corpora is partly ideological, partly practical. Researchers will normally divide the corpus which they have available for development into a training corpus and a test corpus. As both come from the same source the application of the test corpus should give a good impression of whether training the statistical system on the training corpus has achieved a certain level of quality of output.

The main argument for using test corpora, however, is the fact that corpora are more representative of NLP input in the sense that they are real language texts rather than artificially constructed data. The argument is that test suites are far too theoretical in the sense that they collect as many linguistic phenomena as can be found, however obscure they may be. The test suite therefore ends up showing a collection of phenomena that aims to be exhaustive, but which includes many phenomena that are strange and hardly ever (or never) used in real language text. An additional criticism is that test suites are semantically (and pragmatically) uninteresting.

The response of statistical hardliners therefore is to just collect real language texts and analyse them. If a certain phenomenon is important then it will occur with some regularity, and it will need to be implemented and tested. If a phenomenon does not occur in millions of sentences, then they prefer to assume that it does not really exist and to concentrate on the more common things which cause problems.

Note that it is not clear how an evaluator of a statistical system could decide whether the probability information incorporated in a certain system coincide with the statistical information which can be obtained from his own corpus, a set of typical input for which the system has not been trained specifically. In other words, if you take a probabilistic parser to deal with your day to day input texts, then the probabilities may not coincide with the ones required by your texts, because the parser has been trained on a different type of corpus.[2]

6.1.2 Test Suites and Statistical Systems

Let us suppose the usefulness of the test suite technique would largely depend on whether one could relate statistical information to linguistic phenomena or not. However, this relationship will differ for every single statistical system, and it is difficult to make general observations which apply to all of them. There seem to be two main questions which are relevant to the issue:

1. To what extent are systems truly statistical based (i.e. not a combination of linguistic and statistical rules)?

   The UCREL probabilistic parsing system for example was reported to include non-probabilistic rules which are used in order to predict certain limited features of the parse-tree for a given input sentence, so that not all logically possible ways of drawing trees is pursued [12].

2. What type of linguistic notions are being maintained?

   Linguistic notions can be maintained either directly or, probably in most cases, indirectly: the same structures are maintained, but different types of rules (i.e. statistical) are used in order to achieve this.

The latter point is in fact the key issue. The strength of the test suite is that it tries to systematically include a wide range of syntactic phenomena so that the system's performance is tested on all these phenomena. If we can relate these phenomena to probability figures in the system then testing output provides us with useful information on these probabilities. If certain test items fail, it will still be up to the developer to decide whether it is desirable to train the system further on that type of phenomena, depending on the importance of this phenomenon (and maybe on the effect that training might have on the performance of other, related phenomena). One also needs to establish whether, if a phenomena fails, that means that

[2] This is valid unless probalistic systems are always specifically trained for the environment that they have to work in.

all occurrences of this phenomenon fail. It is not at all clear whether this is the case in statistical systems.

But even if a test suite can provide the evaluator with usable information with regard to the performance of a statistical system, the main objection against the method remains the following: a test suite does not reflect real text and therefore fails to indicate how a system would perform in a real working environment. We will come back to this in the next section.

As for semantics, it remains unclear what degree of semantics is currently required for the assessment of semantic probabilities in this type of system. The statistical approach is fairly young and so far little work seems to have been done in the area of semantics.

Adressing the criticisms above will not just make test suites more suitable for usage in statistical based NLP, but would also enhance the usefulness of test suite for the assessment of rule-based systems. However, whether such improvements would lead to the actual usage of test suites for the assessment of probabilistic systems is not clear. For the moment, it seems fairly unlikely.

7 System output

System output can affect test suite (TS) design in a number of ways. First, the inclusion in the test suite of some indication of the kind of output a system is intended to produce is of benefit in interpreting test results (this should be obvious). Second, the different kinds or classes of output that a system can produce also provide a natural (and perhaps non-obvious) basis for organizing the test suite. For example, in a TS for MT, it might be natural to classify the TS sentences according to some properties of their translations. However, The representation and use of intended output constitute the single most difficult problem in TS design, and the biggest obstacle to the design and creation of really general purpose TSs. In the following, we will explore these issues, beginning with some terminology and clarification.

Systems produce *actual output*, and a test suite may contain an indication of *intended output* (e.g. the sort of output one would ideally expect) for each TS item. One may think of the intended outputs as *targets* which the system is supposed to hit. We say "may contain", but this understates matters – TSs *always* contain this information, though it may be implicit. To see this, consider the very simplest TS – a list of sentences which one expects a system (such as a parser) to accept. Here the intended output is implicit: one intends (hopes, expects) the system to accept each TS item – to respond to each TS item by producing some particular behaviour, such as reaching a non-error state. If the system signals this by replying "yes" or "no", then one clearly has actual output. But even if it does not, one can think of the behaviour associated with achieving a certain state (error or otherwise) as a kind of output.

Thus, leaving aside annotation schemes and other information, one should think of a TS not as a list of test sentences (or other expressions), but as a list of pairs: $< I, T >$, where I is an input, such as a test sentence, and T is a "target" or intended output. The problem for evaluation is to compare the actual output from a system with this intended output (T), and to draw useful conclusions (either diagnostic, or relative to some idea of adequacy).

In general, evaluation of systems is easier or harder in the same proportion that it is easier or harder to define and justify generally acceptable targets, and to establish whether these are hit or missed, or (in the case of misses) to measure the magnitude of the miss in some sensible way.

To make matters more concrete, we will consider TSs for applications where targets more complicated than a simple “yes” would be appropriate. These include the only marginally more complex (i) “yes”/“no” replies (that is, acceptance or rejection of input); (ii) syntactic representations, (iii) sets of syntactic representations, (iv) semantic representations, (iv) entire natural language sentences.

Sentence Recognizer “Recognizers” are devices which take an input sentence and answer “yes” or “no” according to whether the sentence is accepted by their grammar or not (i.e. a parser which does not deliver representations). They are useful here for exemplification (they are naturally not much practical use). A TS for a recognizer might be a set of sentences which it should accept, and a set it should reject. We should think of these sets as a single set of $< sentence, target >$ pairs, where the target represents acceptance or rejection – might be represented as an integer (0 or 1), or a string (“yes/no”), etc. Evaluation consists of submitting each sentence to the system and comparing the actual output with the target. This will be relatively easy. It will also be easy to represent the results numerically and to understand their significance. For example, it will be easy to see if one grammar performs better than another, simply by counting scores. Of course, the reason evaluation is so simple is that the system is so simple.

Speech Recognition A TS consists of pairs $< I, T >$ where I is a (recorded) utterance, and T a representation (e.g. a standard phonetic transcription). It is relatively easy to see if a system produces the intended transcription or not, and if not, how close it is, relatively speaking (e.g. count the characters that differ in the target and actual transcriptions). Of course, this ignores the fact that some mistakes are less serious than others (a system that returns /p/ instead of /b/ may be better than one which returns /g/), and it may not be very informative as a diagnostic – for example, the transcription may ignore useful information that the system has available (about intonation, say). However, the general idea is clear, and not much more complex or problematic than in the case of a recognizer.

Part of Speech Tagger The TS may be a set of untagged textual sentences with their tagged counterparts. Again, comparison of actual and intended output is easy, and it is easy to measure how closely the intended output approximates the target (though again, not all errors may be equally serious).

Spelling Corrector A TS might consist of a collection of sentences, with ‘target’ mistakes which the system should recognize, and ‘target’ corrections which the system should propose. The questions are: (i) whether the mistakes the system detects coincide with those in the TS (i.e. the system does not miss any mistakes, and does not correct anything that does not need correction), and (ii) whether the target corrections are included in those the system suggests. This is clearly more complex than the comparisons so far considered, but quite manageable.

Syntactic Parser A TS might consist of a list of sentences, each with a surface syntactic representation (e.g. a parse tree). At first blush, the question appears to be whether, given a particular TS sentence I, the parser produces the associated representation T. Unfortunately, there are a variety of reasons why this may be too simple:

- the parser produces a number of outputs for I (it makes I ambiguous), the question is whether T is among them (if they are delivered with some ordering, perhaps whether

T is among the earlier representations found, or relatively highly rated in some other way);

- conversely: the parser produces a single output, but the TS associates a set of representations with I;
- both parser and TS associate sets of representations with I – the question is then whether the sets coincide, or overlap, and whether one is a subset of the other;

  These sorts of comparisons over sets can be computationally expensive (and are really impractical by hand in any significant numbers), even if the comparison of any pair of individual representations is easy. Unfortunately, it will often not be easy:
- the TS representation (T) may be based on a different theoretical assumptions from those of the parser: e.g. T is a surface phrase marker, and the system produces LFG-style f-structures. The differences may be larger or smaller:
  - simple differences in labelling (e.g. NP vs. PN), but with isomorporhic label sets;
  - where the TS uses a strictly weaker set of labels (e.g. the parser returns complex labels NP_sing and NP_plur, where the TS uses only NP (or vice versa)
  - differences of structure (different assumptions about the attachment of PPs; parser returns binary branching structures, TS assumes n-ary branching);
  - differences of datastructre (feature structures vs. trees).

Semantic Parser The input (I) may be a sentence, and the representation (T) a semantic representation. Here we have all the possibilities just mentioned, plus the problem of logical (or more generally semantic) equivalence – suppose the TS associates I with a representation equivalent to "if p then q", and the parser provides a representation equivalent to "p or not q", which are logically equivalent.

Machine Translation System Inputs (I) will be source language expression and outputs (T) will be target language expressions. There are the problems already noted, but worse, because the range of possible acceptable translations is so much larger, and the difficulty of establishing similarity between target/intended and actual translations so much worse.

One can see two different problems contributing to the more difficult cases. First, it may be difficult to define a standard form of target which systems can be expected to produce (should parsers produce phrase structure trees, and if so with what labels?). Second, given an actual output and a target output in the same format, there is the problem of deciding if they are the same or sufficiently similar: has the system hit the target? (i.e. is the logical form produced the same as – equivalent to – the target one?).

These problems are really intractable. However, there are some ways in which they may be ameliorated.

Simple Targets Since evaluation is harder when the output becomes more complex, restrict TS to very simple outputs (e.g. yes/no); the problem is that this is not very revealing (e.g. suppose a TS for an MT system contains just "yes/no" as target outputs, essentially all one can tell is whether an input receives any translation at all – this is not very useful).

Generic Outputs Here the idea is to establish a 'standard' format for output, which systems under evaluation are supposed to produce (perhaps in addition to anything else they produce for purposes other than evaluation, so this format simply serves a benchmarking purpose). The Penn TreeBank representation might be an example. There is simply no hope of this succeeding in the NLP field. However, there is one case where this is possible and relatively easy, namely where one is designing a TS for diagnostic evaluation of a particular system under development. Here the generic output can just be the intended output (though even here, one must allow for the possibility that the intended output will change, perhaps radically, as the system is developed).

Manipulate the Output Though system output and test output differ, it may be possible to manipulate one or the other so they conform. For example, to translate the node labels produced by a system to those used in the TS (NP $\mapsto$ PN, NP_plur $\mapsto$ NP_pl, etc.). Of course, in some cases this may pose non-trivial problems (e.g. taking phrase structure trees to LFG f-structures), and even then, it will not completely solve the problem of deciding when an actual output is sufficiently similar to the target (recall the problem of logical equivalence). In general, one would expect it to be easier to manipulate the test output – since this need only be done once, while manipulation of system output would have to be done at every test. However, if one is testing several systems with different output, it may still be easier to convert all outputs to resemble those of the TS.

8 User type

Another important factor in test suite design is the type of user. Different types of users may be interested in the result of test suites:

- system developers
- end users
- consumers of linguistic services
- supervisors and sponsors of MT development and researchers in NLP

These different categories of users will have different evaluation requirements and are likely to interpret test suite output in different ways. We leave to another work package the question of interpretation of results.

The user at whom the test suite is aimed is also likely to affect how the test suite items themselves are annotated. What level of education, linguistic training, and language ability should the test suite annotation be aimed at?

The user may may also affect the coverage and organisation of the test suite.

A test suite geared towards a particular user will typically aim at efficient and controlled testing of selected important phenomena (as opposed to and/or as a supplement to using test texts or corpora). Selection of phenomena will frequently be based on corpus analyses and further be reduced to phenomena known or presumed to be difficult for NLP systems.

Test suites may be ordered into sections which allow to test for the most complex test items first, and only consider simpler items in consecutive sections or separate test suites if results on complex items are negative.

9 Comparing Test Suites with Test Corpora

The main testing tool available to evaluators, in addition to the test suite, is test corpora. We take test corpora to mean real text(s) or parts of texts consistent from a terminological, style or level of language point of view. We do not suggest to replace test suites with test corpora, but suggest that they can be seen as a useful, complementary test, especially for adequacy evaluation or for testing the performance of developing systems when they have achieved a certain robustness.

This section will show what makes test suites and test corpora complementary by discussing their respective characteristics. We will concentrate on topics like their nature, aim, coverage, construction, presentation and reusability.

9.1 General Features

In terms of content, a test corpus is likely to be composed of a real text or part(s) of text(s) thus presenting a kind of semantic - discourse dimension brought by the succession of sentences building together a specific meaning whereas the test suites are more likely to be composed of smaller units whose meanings do not necessarily depend on each other. As a matter of fact, test corpora bring a contextual dimension whereas test suites appear to present a certain degree of data isolation (paragraphs are very rare).

Test corpora are also linked to a certain idea of length and big volume whereas test suites are often associated with the idea of representativeness, thus reducing the volume.

Another important difference between the test suites and test corpora lies in the artificial or real aspect of the content. A test corpus is composed of real text whereas very often, test suites are artificially elaborated and are sometimes based on a linguistic theory, which cannot be the case for test corpora.

9.2 Coverage

Coverage seems an interesting topic when discussing the relationship between our two concepts. Considering test corpora, it seems quite evident that their coverage - in terms of linguistic and extra-linguistic phenomena - cannot go beyond their effective occurrence in the corpus unless it is something more sophisticated than a simple text sample but a collection of different real occurrences of the same phenomena (which we do not think is usually the case in existing test corpora). On the other hand, test suites' coverage is supposed to be more generic ... but also more exhaustive. It appears that the linguistic coverage of test suites is bound to be much more "organised" than that of a test corpus. We feel that the reason for this strict organisation is mainly this need of "in depth coverage" because you cannot claim to test a phenomenon through the building of just one sentence.

To cover a phenomenon "in depth", you have to imagine and reproduce all its different expressions according to different criteria (vocabulary, layout, punctuation, transformations, ...). Thus the enlargement of the phenomenon that occurred once in the test corpora to its more generic representation through the building of several sentences or sequences is a quite interesting example of real relationship between test suites and test corpora.

Concerning the breadth of coverage of test suites and test corpora, it is clear that that of the test suites can always be enlarged through the building of new test sequences whereas that of the test corpora is definitive unless some new texts or parts of texts are added, but there is no guaranty of exhaustivity in a corpus because it is not constructed in a systematic way like test suites.

Once again, in the case of the derivation of test suites from test corpora, you can imagine that the breadth of coverage would be the same whereas the "in depth" coverage would be much more important in the test suites. In this situation, test corpora represent a working basis for the exhaustive and in depth phenomenon description in the test suites. Concerning the extra-linguistic phenomena, it seems that test corpora can present a much more realistic coverage than the test suites through the occurrence of real layout and punctuation problems which are quite rare in the existing test suites.

9.3 Presentation

In terms of presentation, it appears that test suites are not in a very close relationship with test corpora. Because if we consider the organisation of data, test suites really constitute a better candidate than the test corpora. Test suites building gives the opportunity to elaborate a phenomenon classification that allows to identify exactly which test suite covers which phenomenon. The management of data and results thus becomes easier than in a test corpus which presents the phenomenon in a non-classified way making it difficult to exploit the results. The use of annotations for example seems to be much more suited to test-suites than to test corpora because it gives information on what is being tested and, hopefully, on what is the expected result for each tested item. The annotation of a test corpus would necessitate a very complex information model because test corpora are not likely to be built in a systematic way.

9.4 Re-usability

Test corpora being directly composed of real texts, the test results will be relevant for one particular corpus and only for that one. The aim of the test is thus limited to observing the adequacy of the system processing with regards to the test corpus that was selected. Test suites being supposed to be more generic, their aim can be enlarged to diagnostic evaluation depending on what the test suites are composed of and the results one gets are not necessarily linked to a particular corpus.

It is clear that the re-usability is a clear divergence point as far as test suites and test corpora are concerned. The corpus orientation by nature is clearly a re-usability reduction factor. As we already said before, test corpora are relevant for one type of corpus according to the corpus selection criteria adopted and the result can not be used for any other type of texts. The tested items of a test corpus are limited to the phenomena including all the constraints brought by the choice of a specific text (vocabulary, specific syntactic style, ...). The difficulty of annotating test corpora is an additional obstacle to the re-usability of such data except, maybe, for progress evaluation, which necessitates a constant reference.

Because they are bound to be more generic, the re-usability potential of test suites seems to be much higher unless they are very corpus dependent. The use of annotation, the classification of the tested phenomena, the need for "in depth" coverage of the phenomena, etc. are some test suites' characteristics (or at least, aimed at characteristics) that constitute a good basis for re-usability. However, the re- usability of test suites strongly depends on the vocabulary used unless they contain only neutral items in terms of linguistic behaviour (which is quite seldom the case!) and even in this case, the goal of the test is not completely reached because you cannot predict the results you would obtain with another type of vocabulary.

9.4.1 Discussion

It will be obvious that when comparing the test suite technique with test corpora assessment there are arguments for and against both. The latter gives the evaluator a better impression of the system's performance on real texts, the final aim of the product. Test corpora are also more interesting from a lexical and semantic/pragmatic point of view, areas which are normally ignored by test suites. However, test suites have some very strong advantages over test corpora, especially with respect to the following:

- Control over test data:

  Test Suites: Testing of phenomena in isolation (or combination) and control over other parameters possible.

  Test Corpora: Phenomena always in combination with other phenomena

- Systematic coverage of phenomena:

  Test Suites: Systematic variation of parameters possible

  Test Corpora: Variations over phenomena only accidental

- Ungrammatical data:

  Test Suites: Negative data constructed as complement to positive data

  Test Corpora: Negative data only accidental or as performance errors

- Systematic annotations:

  Test Suites: Systematic annotations for expected result

  Test Corpora: So far, at best only phrase structure annotations

These advantages will be fully exploited by the TSNLP project.

10 Limitations of Test Suites and their Effects on Test Suite Design

In this section we want to review some of the issues raised in the previous sections in order to evaluate the type of evaluations that test suites could be suitable for, as well as the question of what type of applications can be naturally tested with test suites. Discussing these issues should serve the purpose of making clearer where the limitations on the usage of test suites are.

We assume that test suites basically consist of the test item itself, which serves as system input, some annotation which models the expected system output, and possibly some additional annotations which describe the test item in more detail.

10.1 Evaluation type

Evaluation minimally consists of running a test (in the broadest sense) on a system and interpreting the test result. Test suites trivially can be used for evaluation in that they can form the basis for the testing phase provided that the relevance of the test examples for the domain or application to be tested is guaranteed. They can support the interpretation of a test result only if the interpretation criteria can be formalized (e.g., in the form of annotations).

Given these basic restrictions on the use of test suites a number of different parameters determining the type of evaluation have to be taken into account. These include among others the purpose of the evaluation (diagnostic, progress, adequacy evaluation), the type of evaluator (user vs. developer), the evaluation basis (test suites vs. corpora), the type of evaluated system (module vs. compound system), the evaluation method (blackbox vs. glassbox evaluation), etc.

At first sight one may be inclined to constrain the usage of test suite to diagnostic evaluation performed by a developer on a single module in a glassbox situation. While this is a quite straightforward prototypical use of test suites, we want to discuss below to what degree test suites could or should be involved in other types of evaluation.

As mentioned above in section 3, these three types of evaluation we consider to stand in a potential inclusion relation, rather than representing disjoint types of evaluation. Moreover, as we saw, user evaluation and developers evaluation cannot simply be identified with adequacy evaluation and diagnostic evaluation respectively, if users are interested in more than just the overall performance of a system and if developers are interesting in producing market products.

Another assumption could be that diagnostic and adequacy evaluation coincide with test suite and corpus based evaluation respectively. This may be correct in the sense that a relevant corpus picked from a specific application domain is more likely to provide a representative sample of the task. However, if a test suite has been designed in an adequate way, taking into account the frequency and relevance of phenomena in the respective domain, it may well be almost as useful as a corresponding corpus.

Another hypothetical correlation is the one between user, adequacy and black box evaluation on the one hand and developer, diagnostic and glass box evaluation on the other. What is commonly understood by blackbox and glassbox evaluation is that in the former case the evaluator does not have access to anything but the input and output of a system, whereas in the latter he can have insight into internal states or generally the flow of information between modules. In fact, it is only at the level of monitoring the flow of information between modules that the term glassbox evaluation does make any sense at all. A developer doing diagnostic evaluation of a module or component is usually as much in a blackbox situation about the internal workings of a component as is a user with respect to an entire system. Doing test suite evaluation, i.e., running an entire sequence of systematic test items, fulfills the purpose of diagnosis for exactly this reason, that developers cannot fully keep track of the internal processes of a module in every detail once it has reached a certain complexity. A developer doing test suite diagnostics does not trace every internal step of a module, but rather tries to locate a potential deficiency of the module by looking at the output pattern that emerges from a series of systematic test items. (Removing a deficiency will of course involve access to the code, but this is neither the purpose of evaluation nor of diagnostics proper.)

What appears to be a more crucial factor is the complexity of the tested system, i.e., whether (in the extreme cases) one is doing system evaluation or component evaluation, which presupposes a restricted glassbox situation. One of the main advantages of test suites, as we have seen, is that they allow the construction of test items on the basis of a systematic paradigmatic variation over certain parameters with other parameters remaining stable. As was just mentioned, this property can be used to detect system internal deviations from an expected behaviour, if there are unexpected irregularities in the input/output relation. This is only possible, however, if the expected output can be fully or partially predicted from the input. As soon as there are parameters influencing the system output which are not controlled by the setup of the test items, at least the automatic and systematic evaluation by means of a test suite becomes less reliable. Such irregularities could arise because of dependencies on extra-linguistic configurations, as for example in a dialogue system, or because the input/output relation is systematic only with respect to some components of the system, while others may distort this systematic relation. For this reason test suites appear most suitable for the testing and evaluation either of components or of systems which behave in a relatively predictable and deterministic manner.

To summarize: Test suites are not only suitable for diagnostic and progress evaluation, but can also support adequacy evaluation, if the evaluator is interested also in more detailed information about deficiencies and customizability of a system. They can complement the function of corpora in evaluation if they are tuned towards the relevant domain, in the sense of containing information about frequency and relevance. They are suitable for blackbox evaluation under condition that the systematic paradigmatic patterns in the test suite in correlation with the expected system output allow to make inferences about internal irregularities of a system. This presupposes that the parameters influencing the output of a system or component are controlled or systematically dependent on the input test item.

10.2 System/Application/Component Type

Much of what has been said in the previous section is of immediate relevance for determining the type of application or system that can be appropriately tested by means of test suites. To answer this question we have to be aware of what kind of information can be encoded in a test suite.

A most basic distinction associated with a test item is whether it belongs to a language or not. This distinction may coincide with the notion of grammaticality, if the language in question is a natural language, or it may be based on an artificial distinction cutting out a sublanguage in the context of a controlled language checker.

Since a test suite cannot enumerate all sentences of a language, it has to simplify certain phenomena, such as recursion, it may have to generalize over certain phenomena, such as lexical distinctions, and it may have to abstract from possible combinations of phenomena occurring in natural language, such that an appropriate degree of minimality is achieved.

With respect to the issue of phenomena, the question is how they are chosen and whether a specific phenomena is accounted for by just one or two examples or by means of several items representing a minimal set. This will correspond by and large to testing the breadth and depth of coverage of an application.

Given these basic constraints, the question is how the information is conveyed in the test suite. Basically, information can be carried either by the test item or by a sequence of test items forming a systematic pattern or it can be provided by annotations to the test item.

The preferred, or at least more reliable, way of encoding information should always be the former, since no matter how annotations are chosen they will either have to be specific to an application or even to a system, or they will have to be translated to be matched with the system's output. If it is possible on the other hand to structure test items such that the patterns they form already encode certain information, the problem of localizing a test suite for a specific system will not arise.

Clearly, in many cases the information about the expected output will have to go into the annotations. However, what has to be taken into consideration here is that such annotations only make sense if they can be matched with the system output. This will be possible only if the range of possible output can be narrowed down to be predictable from the input. Whether or not this is possible will strongly depend on the kind of application chosen as well as on which parts of an application are to be tested.

If we consider for example the most basic annotation of grammaticality with respect to some (sub-)language, there are at least two types of applications which crucially draw on this kind of information, namely grammar checkers and controlled language checkers. What can be tested in these applications is basically whether they accept all grammatical sentences (or all sentences belonging to a sub-language) and reject or mark all ungrammatical sentences. If the kind of error report expected from the system in the case of ungrammatical sentences is known, this information could also be encoded in the annotations in a customization phase. The same may hold for suggestions for corrections, if there is a fixed protocol for every error. To account for the error report and corrections will be best done in isolation or controlled combination, i.e., the occurence of errors test items should be as minimal as possible.

The information about grammaticality will not play any role, however, if we consider other applications, such as for example MT systems or in fact any system which involves parsing. The reason is simply that these systems usually ignore exactly this kind of information in order to achieve a certain robustness. I.e., even if somewhere internal to the system there may have been a stage where a sentence was classified as ungrammatical, any respectable system will not just report a parse failure but try to recover and produce some output.

As we discussed in section 4 MT systems provide a typical example of where it may be rather difficult to make predictions about the system output, mainly because this output is canonized only to a limited degree and subject to many stylistic variations. Again, what may be tested by means of a test suite are minimal constructions for which the translations are expected to follow a certain common standard.

To summarize: We have to assume that test suites will have to abstract from quite a number of factors occuring in natural language in order to be maintainable and at the same time informative. Whether or not an application can be automatically or semi-automatically tested with a test suite will very much depend on the predictability of the output and the range of variations to be expected. In many cases this will require that test items are as minimal as possible, which could mean that they are less informative since the value of some application may crucially depend on how it handles combinations of phenomena. On the other hand, with the number of phenomena interfering within a test item increasing, it is to be expected that the predictability of the output decreases, such that the output control by the human evaluator is required again.

References

[1] H. Alshawi, D. Carter, B. Gambäck, and R. Mayner, *Translation by Quasi Logical Form Transfer*, in Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics, pp161-168, University of California, Berkeley, California, July 1991.

[2] H. Alshawi (ed), *The Core Language Engine*, Cambridge, MIT Press, 1992.

[3] L. Balkan, A. Bech, S. Douglas, S. Manzi, B. Maegaard, C. Mazzi, *Writers' Aids: Interim Report*

[4] E. Black, others (1992), title, in: *Proceedings of ACL.*

[5] Shona Douglas, *Intelligent Text Processing: A Survey of the Available Products* University of Edinburgh, June 1990.

[6] Draft Interim Report EAGLES Evaluation Group, 1994.

[7] Estival, Falkedal et al. (1994), *Analysis of Existing Test Suites*, Report to LRE 62-089 (D-WP1), University of Essex.

[8] S. Flank, A. Temin, H. Blejer, A. Kehler, S. Greenstein, *Module-level Testing:Lessons Learned*, in J.G. Neal and S.M. Walter (eds), *Natural Language Processing Systems Evaluation Workshop*, Report RL-TR-91-362, Rome Laboratory, 1991.

[9] Flickinger, D., J. Nerbonne, I.A. Sag & T. Wasow (1987). "Toward Evaluation of NLP Systems". Hewlett-Packard Laboratories, distr. at the 24th Annual Meeting of the Association for Computational Linguistics (ACL), Stanford.

[10] J.R. Galliers and K. Sparck Jones, *Evaluating Natural Language Processing Systems.* University of Cambridge, England, Technical Report No. 291, 1993.

[11] B. Gambäck, *Developer Oriented Evaluation: Two Experiments*, talk presented at the Workshop on the Strategic Role of Evaluation in Natural Language Processing and Speech Technology, Record of the ESPRIT DANDI/ELSNET/HCRC Workshop, Edinburgh, 1992.

[12] R. Garside, G. Leech, G. Sampson (ed.), The Computational Analysis of English — A Corpus-Based Approach, Longman, Harlow, 1987.

[13] A R. Grishman, A C. Macleod and A J. Sterling, *Evaluating parsing strategies using standardized parse files*, in Proceedings of the 3rd Conference on Applied Natural Language Processing, Trento, 1992

[14] Harrison et al., *Evaluating Syntax Performance of Parser/Grammars of English*, in Jeannette G. Neal and Sharon M. Walter, *Natural Language Processing Systems Evaluation Workshop*, Calspan-UB Research Center, 1991.

[15] Heid, Ulrich & Elke Hildenbrand. (19..). "Some practical experience with the use of test suites for the evaluation of SYSTRAN."

[16] James Hoard, *Preliminaries to the Development of Evaluation Metrics for Natural Language Semantic and Pragmatic Analysis Systems*, in J.G. Neal and S.M. Walter (eds), *Natural Language Processing Systems Evaluation Workshop*, Report RL-TR-91-362, Rome Laboratory, 1991.

[17] King, M. (1988) "A Practical Guide to the Evaluation of Machine Translation Systems". Issco, Geneva. Interim Report to Suissetra

[18] King, M. & K. Falkedal (1990). "Using Test Suites in the Evaluation of Machine Translation Systems". *Proceedings of the 13th International Conference on Computational Linguistics (COLING),* Helsinki.

[19] Judith Klein, John Nerbonne, Klaus Netter, Adbel Kader Diagne, and Ludwig Dickmann, *A Diagnostic Tool for German Syntax*, in Busemann, S., and Harbusch, K. (eds), *Proceedings of the DFKI Workshop on Natural Language Systems: Reusability and Modularity*, October 1992.

[20] Karen Kukich, *Techniques for Automatically Correcting Words in Text* ACM Computing Surveys, Vol. 24, No. 4, December 1992.

[21] John Lehrberger and Laurent Bourbeau, *Machine Translation: Linguistic characteristics of MT systems and general methodology of evaluation*, John Betjamins, 1988.

[22] Marcus, Mitchell P., Mary Ann Marcinkiewicz, Beatrice Santorini (1993): "Building a Large Annotated Corpus of English: The Penn Treebank", *Computational Linguistics*, 19(2), pp.313-330.

[23] Neal et al. (1992), An Evaluation Methodology for Natural Language Processing Systems, RL-TR-92-308, Final Technical Report, Rome Laboratory, Air Force Material Command, Griffiss Air Force Base, New York.

[24] J. Nerbonne, K. Netter, A Kader Diagne, J. Klein, L. Dickman, *A Diagnostic Tool for German Syntax.* DFKI D-92-03, Saarbrücken. Also in: J. Neal / S. Walter. eds. (1991): *Natural Language Processing Systems Evaluation Workshop, Berkeley.* Rome Laboratory, RL-TR-91-362, New York

[25] J. Nerbonne, *Evaluating Semantic Modules for Natural Language Understanding*, in H. Thompson (comp), *The Strategic Role of Evaluation in Natural Language Processing and Speech Technology*, Record of the ESPRIT DANDI/ELSNET/HCRC Workshop, Edinburgh, 1992. TSNLP-WP2-SX-0

[26] Nübel, Rita. (1993). "Möglichkeiten zur Evaluierung eines kommerziellen MU-Systems: Erfahrungsbericht aus computerlinguistischer Sicht". IAI, Universität des Saarlandes.

[27] Martha Palmer and Tim Finin, *Workshop on the Evaluation of Natural Language Processing Systems*, Computational Linguistics, V 16, No 3, Sept. 1990.

[28] W. Read, A. Quilici, J. Reeves, J. Dyer, E. Baker, *Evaluating Natural Language Systems: A Sourcebook Approach*, Coling 88, pp 530-534.

[29] S. Regnier, E. Dauphin, *Test Suite Design at Aerospatiale*, Input Paper for WP2, TSNLP.

[30] Richard H. Wojcik, Philip Harrison, John Bremer (1993), Using Bracketed Parses to Evaluate a Grammar Checking Application, in: *Proceedings of ACL*, Boeing Computer Services Research and Technology Division, Seattle.