Abstract

Development and evaluation of spoken language dialogue interfaces (SDIs) still suffers from lack of accepted standards or widely understood benchmarks for assuring potential customers and users of the quality of systems and components. The DISC project has been addressing this problem by drafting a best-practice dialogue engineering methodology and then testing that methodology. This chapter presents ongoing work in DISC on evaluation of SDIs and their components. A proposal is presented for (i) how to systematically generate a comprehensive set of evaluation criteria for SDIs and their components, and (ii) how to thoroughly characterize each evaluation criterion through the use of a common template describing the what, the when, the how, the importance, the difficulty, the cost, etc. of evaluation. The approach is illustrated by the case of dialogue manager evaluation.

• Introduction

Spoken dialogue interfaces (SDIs) have within the last few years begun to attract broad industrial interest. This is primarily due to the advent of sufficiently robust, speaker independent, continuous speech recognizers rather than to particular developments with respect to the integrated SDIs themselves, which are in a gradual process towards maturity (Bernsen et al. 1998a). However, SDIs development and evaluation still suffer from the lack of accepted standards or widely understood benchmarks for assuring potential customers or users of the quality of particular applications. Similarly, there are no reliable methods for comparing the quality of two SDIs before selecting one for deployment in the field. Needless to say, this situation continues to generate uncertainty about the potential of SDI technologies, their proper domains of application, their usability, the cost of producing them, their development time and the quality of products in both absolute and comparative terms. In an increasingly competitive marketplace, the ability to state that some system has been developed following a carefully designed and validated dialogue engineering methodology, along with the ability to report evaluation results in a standardised framework, will give products developed in this way a competitive advantage. And that, in turn, is likely further to stimulate adoption of the methodology as well as of the technology itself.

The European Esprit Long-Term Research Concerted Actions DISC (Spoken Language Dialogue Systems and Components. Best practice in development and evaluation) and its successor, DISC-2 are developing

a detailed and integrated set of development and evaluation methods, procedures and tools which are intended as a first dialogue engineering best practice methodology. DISC focuses on six key aspects of SDIs: speech recognition, speech synthesis, language understanding and generation, dialogue management, human factors and systems integration.

The DISC approach has been to advance towards a first definition of best practice through a thorough investigation of current dialogue engineering practice for the development and evaluation of SDIs and their components. The current pilot investigation has been executed through analyzing a broad range of SDIs and components with respect to the key aspects mentioned above, and mapping out their respective development and evaluation processes. In order to adequately capture current practice and overcome various problems primarily relating to the insufficient and incomparable information provided for individual systems and components, a common scheme has been developed through several iterations. The scheme consists of a 'grid' and a life cycle model both of which are slot-filler structures based on current knowledge of SDIs and of software engineering life cycles, respectively. At the time of writing, DISC-2 is about to start and the DISC dialogue engineering scheme has reached a draft best-practice stage.

This chapter briefly presents the DISC dialogue engineering scheme as a basis for addressing some key issues of SDI evaluation. Section 2 presents the DISC grid and life cycle. Section 3 presents the DISC terminology for various types of evaluation and test. Section 4 presents a general approach to the evaluation of SDIs and components including a template for characterizing evaluation criteria. Section 5 discusses application of the approach to the evaluation of dialogue managers. Section 6 concludes the chapter.

• The DISC Dialogue Engineering Scheme

Development and evaluation are tightly interwoven and should interact closely throughout the process of creating an SDI or component. The DISC life cycle model is intended to capture all such process-oriented issues whereas the grid describes the properties or characteristics of the SDI or component being developed and evaluated (Heid et al. 1998).

• The DISC grid

The DISC grid takes the form of a series of "checklist" entries that should enable a comprehensive and in-depth characterization of the

properties of any SDI or SDI component. The first DISC grid was heavily based on the interactive speech theory illustrated in Figure 1 (Bernsen et al. 1998a) and was somewhat biased towards dialogue management and human factors, i.e. the performance, control and context layers in Figure 1. Subsequent versions of the grid have been substantially expanded, particularly as regards the language and speech layers, during the current practice investigation described in Section 1.

<insert Figure 1 here>

The slots of the final DISC grid cover SDI component architecture and function, system architecture and issues of system integration, multimodality and general system performance, as well as properties of individual components, including speech recognition and speech synthesis, language processing for user and system utterances, and dialogue management.

- **The DISC life cycle**

The DISC life cycle model aims to capture the development and evaluation process for SDIs and their components. The first model was based on work presented in (Bernsen et al. 1998a) and has subsequently been extended as a result of project discussions and the current practice investigation described in Section 1.

Figure 2 (Bernsen et al. 1998a) shows a general software engineering life cycle model which has been slightly specialised to the development and evaluation of SDIs and their components. The figure presents an overall framework for the development and evaluation process through to installation at the user's site. After this point may follow software maintenance, the software may be ported to other platforms or may be adapted to new applications in which case a new life cycle begins.

<insert Figure 2 here>

Drawing on a general software engineering life cycle model, the DISC dialogue engineering life cycle model is specialised to capturing the process of developing and evaluating SDIs and components. The model includes a series of fairly detailed questions which address overall design goals as well as constraints on, and resources of, the development and evaluation process, such as user and/or developer preferences, time, money, people available for development and evaluation. Attention is paid to the availability of process documentation at all stages, as well as to the way in which the major engineering issues, such as robustness, maintenance, and portability, are handled.

The DISC life cycle model is intended to be used for describing an entire SDI as well as a single component. This is possible because each piece of software has a life cycle and most life cycle issues are relevant independently of the specific nature of the software. The exact nature of the questions asked on each process issue may, however, depend on the particular nature of the software. For instance, the issue of evaluation is highly relevant to, e.g., speech recognizers as well as dialogue managers. However, the evaluation criteria involved are very different.

In the following we focus on the central evaluation entries in the DISC life cycle model, i.e. those concerning types of evaluation and test (Section 3) and evaluation criteria (Sections 4 and 5). In fact, many other entries in the DISC life cycle model address evaluation to a greater or lesser extent as well, because of the close relation between evaluation and development. However, space limitations allows us to discuss the core evaluation entries considered in DISC. More information will be available at http://www.elsnet.org/disc/ in due course.

• Types of Evaluation and Test

During the dialogue engineering life cycle, evaluation is constantly needed for measuring progress towards the goals which the SDI or component must meet. However, evaluation of individual SDI aspects as well as of entire SDIs is today as much of an art and a craft as it is an exact science with established standards and procedures for good engineering practice. There is not even consensus on terminology in the field.

Distinction may be made among three types of evaluation ((Bernsen et al. 1998a), see also (Hirschman and Thompson 1996) and (Gibbon et al. 1997)). Although clearly not orthogonal, these three types seem to cover the relevant aspects of evaluation and subsume the scopes of other commonly used terms and distinctions. Each type may be used at any stage during SDI or component development:

- performance evaluation, i.e. measurements of the performance of the SDI or component and its modules in terms of a set of quantitative and/or qualitative parameters;
- diagnostic evaluation, i.e. detection and diagnosis of design and implementation errors;
- adequacy evaluation, i.e., how well does the system or component fit its purpose and meet actual user needs and expectations?

Other common terms are 'blackbox' and 'glassbox' tests, and 'progress evaluation'. Blackbox and glassbox tests may be considered forms of diagnostic evaluation but these tests are carried out on implemented components or systems only (see elaboration below). Progress evaluation is used to compare two iterations of the same system or component during development, such as two Wizard of Oz iterations of an SDI.

Performance, diagnostic and adequacy evaluation should be performed as integral parts of the development process to measure progress towards satisfaction of the requirements specification, evaluation criteria and design specification.

Performance evaluation is made throughout the development process with approximately the same emphasis between iterations.

Diagnostic evaluation is of central importance in the early development process but should require less effort in the final phase by which time most errors should have been removed. During debugging of the implemented SDI or component, two typical types of test are glassbox tests and blackbox tests. There is no general agreement on the definitions of glassbox and blackbox tests. Here we will use *glassbox test* to indicate a test in which the internal system representation can be inspected. The evaluator should ensure that reasonable test suites, i.e. data sets, can be constructed that will activate all loops and conditions of the program being tested.

In a *blackbox test* only input to and output from the program are available to the evaluator. Test suites are constructed in accordance with the requirements specification and along with a specification of the expected output. Expected and actual output are compared and any deviations must be explained. Either there is a bug in the program or the expected output was incorrect. Bugs must be corrected and the test run again. The test suites should include fully acceptable input as well as borderline cases to test whether* the program reacts reasonably and does not break down in case of errors in the input. Ideally, and in contrast to the glassbox test suites, the blackbox test suites should not be constructed by the programmer who implemented the system since s/he may have difficulties in viewing the program as a black box.

Adequacy evaluation of SDIs and components typically includes some general performance measurements as well as measurement of user satisfaction. Adequacy evaluation is used mostly in the later phases of development. This is because a number of adequacy aspects cannot be tested in a sensible way until an implemented and debugged SDI or component is available for the purpose. For instance, it does not

make sense to measure real-time performance on a simulated system. The evaluation of user satisfaction of individual SDI components raises several important difficulties. Sometimes, it can be difficult or impossible to do. For instance, it is probably impossible to evaluate user satisfaction with respect to deeply embedded components, such as parsers. Furthermore, user satisfaction of individual SDI components is non-transitive in an important sense: it is possible to build an unsatisfactory (to its users) SDI from components which are individually satisfactory to users in so far as this can be evaluated.

Other useful distinctions are those between quantitative and qualitative evaluation and subjective and objective evaluation. *Quantitative evaluation* consists in counting something and producing an independently meaningful number. It should be noted that, even if quantitative measures may make little sense in absolute they can be useful for progress evaluation in which improvements are being measured. However, we would argue that progress evaluation is not to be considered quantitative evaluation unless progress is measured against an independently meaningful quantitative standard or target. Independently meaningful scores are important for purposes of comparative evaluation of systems and components, but they are difficult to achieve. For instance, many published speech recognition success rates suffer from underspecification in terms of factors such as recording environment, microphone quality, corpus selection, corpus size, speaker population details etc.

Qualitative evaluation consists in estimating or judging some property by reference to expert standards and rules.

Subjective evaluation consists in judging some property by reference to users' opinions.

Objective evaluation addresses objectively measurable performance parameters. Performance evaluation and diagnostic evaluation are forms of objective evaluation whereas adequacy evaluation includes both objective and subjective evaluation. Both quantitative and qualitative evaluations are objective evaluations.

In addition to distinctions between different types of evaluation such as the above, it is useful to distinguish between different types of test. Test types differ with respect to certain aspects of the context of the evaluation, such as the users involved, whether or not scenarios are used, and whether the system being tested is an implementation or a simulation. We distinguish between controlled tests, field tests and acceptance tests. Roughly speaking, controlled tests are performed during simulation and after implementation; field tests are performed

after implementation and towards the end of systems development; and the acceptance test is the final test of a system. Each test typically includes performance, diagnostic, and adequacy evaluation.

In a *controlled test,* the users need not be those who will actually use the final system. However, it is recommended that one select the test subjects from the target user group to ensure that they have relevant backgrounds, professional and otherwise. In a controlled test, the tasks to be carried out (the scenarios) should not be selected by the participants. To ensure scenarios that are reasonably representative with respect to system functionality and task domain coverage, and to bring the controlled test as close to benchmarking as possible, scenario selection should ideally be done by an independent panel according to guidelines on, for example, who should select the scenarios, their coverage of system functionality and task domain, the number of scenarios per user and the number of users. The panel should include end users as well as system developers. A *field distribution problem* attaches to all results of controlled tests. The frequency of different tasks across the domain of application may be different in real life from that imposed in the controlled test. This may significantly affect the frequency of the interaction problems encountered in the test.

In a *field test,* the SDI or component is being tested by real end users in their appropriate environments. This means that the experimental tasks will correspond to real-life tasks but may, nonetheless, fail to be representative of the full range of system functionality unless the duration of the field test is very long. The field test option will not always be available for research systems due to the absence of a real customer. It may be preferable to carry out a controlled test before the field test because the controlled test will allow an evaluation that is close to benchmarking.

The *acceptance test* is the final test of the SDI or component before it is accepted for operational use (Sommerville 1992). The test aims to demonstrate that the contractual requirements (the requirements specification) and evaluation criteria have been satisfied. Often the SDI or component is tested with data supplied by the procurer or in a set-up specified by the procurer. Detected errors must be corrected immediately. In case of larger disagreements with, or omissions in, the requirements specification, developer and procurer must discuss what to do. In the worst case the procurer may turn down the product if the component or system does not meet the requirements agreed upon. However, it is not always solely the system developer's fault that the SDI or component does not exhibit the performance and functionality

anticipated by the procurer. In such cases, procurer and developer must negotiate a resolution.

- **Evaluation Criteria**

Before applying the types of evaluation and test described in Section 3, we need to know what could, or should, be evaluated in a particular SDI or component, i.e. we need a set of evaluation criteria. This is where research into evaluation of SDIs and components comes into its own with no further support from general software engineering best practice. In what follows, we would like to argue in favour of two basic points. Firstly, an evaluation criterion is a complex entity that should be characterized as such. Secondly, for any SDI or component there is an important issue of completeness with respect to which evaluation criteria could be applied to that SDI or component. To become an applied science, dialogue engineering needs to have explicitly defined sets of evaluation criteria from among which to select, that are reasonably complete from a state-of-the-art point of view. And each possible evaluation criterion must have a comprehensive description. For the moment, is seems that the field is lacking in both respects, often making do with a small set of evaluation criteria selected *ad hoc* together with incomplete characterizations of the selected criteria. DISC is committed to a systematic approach to evaluation of SDIs and their components, as follows.

Based on a partially ordered list of possible properties of any particular SDI or component as expressed in the DISC grid, a systematic and comprehensive overview can be generated of the evaluation criteria that may be used for the evaluation of that SDI or component. For each property characterizing a particular SDI or component, the question for evaluation is, roughly: is the property adequate or not? In other words, once we know what the core properties are, we have a basic grasp of what could be evaluated in SDIs and their components.

However, knowing what could be evaluated in a system is far from knowing *how* to assess a particular property, *when* in the software life cycle to do it, which *type* of evaluation one is dealing with, etc. Such questions can be asked with respect to any property and its corresponding evaluation criterion. This leads to the idea of creating a standard *evaluation template* which explains the things one needs to know about a particular evaluation criterion (i.e. the when, the how etc.) in order to correctly apply the criterion. The evaluation template itself is a generic construct whose appropriateness is presently being

tested on the dialogue manager component of SDIs (Section 5). The working hypothesis is that the template will turn out to be applicable not only to dialogue manager evaluation but to evaluation of all aspects of SDIs and their components.

The template includes seven entries and draws upon the terminology explained in Section 3. The template is a generic tool that must be filled in for each property to be evaluated. In the operational version of the template, the terminology needed is explained in the template itself. The generic entries are:

A. What is being evaluated

Describes the *property or properties* of an SDI or component being evaluated, such as speech recognition success rate. In some cases, an evaluation criterion refers to a generic property that covers several specific properties. Dialogue segmentation, for instance, can be done in several different ways depending on the segmentation units involved, such as user and system turns, or dialogue acts. In these cases, the evaluators using the template will have to make the appropriate specifications of the particular properties that they will be evaluating.

B. System part evaluated and type of evaluation

Describes (i) the *system part* that is being evaluated, i.e. if what is being evaluated is an SDI as a whole, an SDI module, such as the speech synthesizer, a sub-module, such as a particuler dialogue history, or several modules or sub-modules; and (ii) the *type* of the evaluation, i.e. whether evaluation is quantitative, qualitative or subjective.

C. Method(s) of evaluation

Describes which methods of evaluation may be used at various stages in the life cycle. In early design and specification, evaluation tends to be conceptual rather than based on real data. Later in the life cycle, data capture and analysis dominate the evaluator's activities (see generic template entry E below).

Design analysis consists in using experience and common sense, and thinking hard when exploring the design space during the specification and design phases, doing walkthroughs of models, comparing with similar systems, browsing the literature, applying existing theory and guidelines, if any, involving experts and future users, the procurer etc. The completeness of the requirements specification may be judged by checking whether all relevant entries in the DISC grid have been covered. Evaluation at this stage also consists in checking whether goals and constraints are sound, non-contradictory and feasible given the resources available.

Wizard of Oz data analysis consists in analyzing problems posed by phenomena observed in data from simulated user-system interactions. The simulations are performed by one or several humans and address the unimplemented parts of the system. These may range from the entire system to a single sub-module, such as a fully implemented system in which only the recognizer is switched off and replaced by simulation. The advantage of simulations is that, if done extensively and analysed carefully, a large number of problems with design concepts and the phenomena that will be present in the deployed application can be spotted before implementation begins. Their disadvantage is the cost of setting up and running several simulations, and the subsequent cost of analyzing the generated data.

Running I/O test suites in 'blackbox' and 'glassbox' evaluation during the implementation phase (see Section 3).

User-system interaction data analysis consists in analysis of data from the interaction between the fully implemented system and real users, either in controlled experiments with selected users and scenarios that they must perform, or in field studies where nothing is under the control of the developers. User-system interaction data is useful or even essential when too little is known in advance about ~~the~~ how users react to characteristics of the deployed application. This tends to be reliable if from a test corpus of sufficient size and realism with respect to task and user behavior. Unfortunately, the data cannot be obtained until late in the development of the system and collection is costly. In addition, this data is often hard to analyze because it can be multiply ambiguous with respect to the cause of some observed problem.

D. Needs and dependencies

Comments on the *need for the property* being evaluated, which may be *relative* to other factors that are specified, such as the task or the distribution of dialogue initiative among user and system.

E. Life cycle phase(s)

Describes the life cycle phases in which evaluation of the property in question should be performed (Figure 2). In general, the earlier evaluation can start, the better. Distinction is made between early design, simulation, implementation, field evaluation and final evaluation.

Early design including requirements and design specification. This is the most important life cycle phase for system and component evaluation. However difficult this may be to do in any formal way, it is essential to carry out a systematic, explicit evaluation of whether the design goals and constraints are reasonable, feasible and non-

contradictory. Caught at this stage, errors due to rash design decisions will not be causing trouble later on. There is no better substitute for qualitative evaluation and sound judgement during early design. This also explains the importance of developing applied theory, guidelines and tools in support of early design. These support mechanisms should help developers know what to look for in the evolving design specification.

Simulation and implementation. This is the life cycle phase in which modules, such as the dialogue manager and its sub-modules, should be strictly tested. To begin with (part of) the SDI or component may be simulated while the end result of this phase should be an implemented and debugged system or component ready for external trials. Simulation-before implementation may be advisable in many cases, not least with respect to dialogue manager development. Applied theory and guidelines are at this stage mainly used in support of scenario and test suite development.

Field evaluation is performed by exposing the SDI or component to uncontrolled interaction with users. Most properties of system components, such as dialogue managers, are difficult to evaluate at this stage. Field evaluation may precede the final acceptance test (Figure 2).

Final evaluation may consist in an acceptance test, i.e., a formal and controlled evaluation which should decide if the system, such as an SDI, meets the evaluation criteria specified as part of the requirements specification. What is primarily being evaluated is the behavior of the system as a whole.

F. Importance of evaluation

Assesses the importance of evaluating individual properties. Note that importance is a multi-faceted concept and may depend on, among other things:

- is evaluation of this property *relevant to all or only some* current systems or components?
- does the system or component have the property under consideration, *how crucial* is it to get the property right? What are the penalties?

Evaluation importance can be described as *low, medium* or *high* together with a statement of the reasons for the grading.

G. Difficulty and cost of evaluation

Assesses the difficulties and costs involved in performing the evaluation:

- the difficulty of evaluation may depend on various forms of *complexity*, such as task complexity, user input complexity, dialogue manager complexity, or overall system complexity;
- the difficulty of evaluation may depend on the existence of *unsolved research problems*. These may be more or less severe;
- evaluation is more or less *costly* to perform in terms of time, manpower, or skilled labour.

- **Evaluation Criteria for Dialogue Managers**

To each SDI aspect, corresponds a set of evaluation criteria. In this section we illustrate the DISC approach to evaluation through the aspect of dialogue management. The current state of practice for dialogue manager evaluation is less mature than other SDI component evaluation, but this makes dialogue manager evaluation an interesting testing ground for the DISC approach to evaluation.

In addressing the issue of dialogue manager evaluation, it is important to keep two different situations in mind, i.e. (1) evaluation of the SDI of which the dialogue manager forms a part, and (2) evaluation of the dialogue manager *per se*. Dialogue manager evaluation is required in both situations. However, (1) obviously makes it harder to distinguish between those parts of the SDI's performance which are due to the dialogue manager alone, those which are due to the performance of other system components, and those which are due to interaction between the dialogue manager and other system components. Poor speech recognition, for instance, can only to a certain extent be counter-balanced by good dialogue manager design. If the speech recognition is too poor, the users will walk away even if they are faced with a brilliant dialogue manager. (2) may be one in which the dialogue manager is being selectively evaluated as part of SDI development or it may be one in which the dialogue manager is itself the sole target of development. In both of the latter cases, the evaluation criteria involved are likely to be more particular to a given dialogue manager than those involved in evaluating the dialogue manager as part of an SDI as a whole. As the dialogue manager influences many different parts of SDI performance and processing, a full list of dialogue manager evaluation criteria is likely to overlap with evaluation criteria for system integration as well as evaluation criteria for the human factors aspect of the system. This is not a problem in DISC which covers all these aspects of SDIs but it does imply some vagueness with respect to whether or not a certain evaluation criterion pertains to dialogue management.

Only a few years ago, the field of dialogue management was so new that evaluation criteria hardly existed at all, i.e. no one had really thought about what to test, how to test etc., and no experience from previous development efforts was available. Evaluation criteria were invented *ad hoc* when the dialogue manager and the SDI in which it was embedded came up for evaluation. This way of doing things is still quite common, in particular in research projects, but cannot be recommended because it means that developers have little support during development in terms of criteria that the dialogue manager should fulfil in order to be considered satisfactory and acceptable. The definition, from the start of the life cycle, of clear, relevant and appropriate evaluation criteria, and the continuous and methodologically sound evaluation of progress with reference to those criteria, should be main characteristics of dialogue manager development and evaluation.

There is still no well-defined set of evaluation criteria to draw upon in the literature. Most criteria in current use are purely quantitative, such as transaction success (Bernsen et al. 1998a). These can be both useful and relatively easy to apply but provide insufficient information on the real quality of a dialogue manager. Subjective measures from user questionnaires and the like, on the other hand, such as scorings of perceptions of an SDI on a five-point scale, are often difficult to interpret as input on the quality of the dialogue manager. Objective qualitative evaluations from experts are difficult to come by already because there are so few experts in this field. In this situation, it seems that one thing that might help advance the state-of-the-art is to generate all possible evaluation criteria for dialogue managers, represent each criterion using the template presented in Section 4, and test how this apparatus works in real development projects in industry and research.

Based on in-depth analysis of a series of dialogue managers, a semi-structured list of possible dialogue manager properties was established. The list (see below) follows a model of significant possible steps in the dialogue manager's processing of input (Bernsen et al. 1998b). Each dialogue manager property encountered on the list is a possible "what" for evaluation (cf. Section 4, template entry A) and must be described according to the evaluation template presented in Section 4. Somewhat to our surprise, the list of possible dialogue manager properties generated no less than 37 possible evaluation criteria for dialogue managers. The criteria are presented in a structured list that mostly follows the model of significant possible

steps in the dialogue manager's processing of input just referred to, as follows:

1. Use of knowledge of the current dialogue context and local and global focus of attention. 3 evaluation criteria. Example: dialogue segmentation adequacy.
2. Map from the semantically significant units in the user's most recent input (if any), as conveyed by the speech and language layers, onto the sub-task(s) (if any) addressed by the user. 5 evaluation criteria. Example: sub-task or topic identification success.
3. Analyse the user's specific sub-task contribution(s) (if any) through the execution of a series of preparatory actions (consistency checking, input verification, input completion, history checking, database retrieval, etc.). 4 evaluation criteria. Example: database information sufficiency.
4. The generation of output to the user, either by the dialogue manager itself or through output language and/or speech layers and/or other output modalities. 10 evaluation criteria. Example: adequacy of on-line information to users on how to interact with the system.
5. Change or update its representation of the current dialogue context, and generate whatever constraint-based support it may provide to the speech and language layers. 1 evaluation criterion. Example: adequacy of dialogue manager support for the speech and/or language layers.
6. Global issues of dialogue management evaluation. 7 evaluation criteria. Example: feedback strategy sufficiency: processing feedback.
7. Global issues of dialogue system evaluation. 7 evaluation criteria. Example: real-time performance.

It is only the criteria belonging to (6) and (7) above which do not correspond to the processing model, illustrating the point made earlier in this section that some dialogue manager evaluation criteria come close to being evaluation criteria for SDIs as a whole.

It seems obvious that such a long list of evaluation criteria for dialogue managers is potentially counter-productive. We might be seen as arguing that, rather than relying on a small and relatively arbitrary set of dialogue manager evaluation criteria as is currently the case, developers should start spending very considerable resources in order to evaluate their dialogue managers from 37 different points of view whenever appropriate throughout the life cycle. What we are

recommending is something else, however. Firstly, no existing dialogue manager has all the *possible* properties listed in the DISC dialogue manager processing model. For instance, no current commercial system, seems to be using indirect speech act recognition. So their developers need not worry about getting indirect speech act recognition right. Secondly, the template presented in Section 4 contains entries that address the importance of evaluation as well as its difficulty and cost. The purpose of these entries is to assist developers in focusing their evaluation efforts on the most important properties of their dialogue manager, including those whose evaluation targets should be included in the requirements specification. We hope that these entries, when properly defined for each possible dialogue manager property, will provide helpful guidelines for the focusing process. Still, we are dealing with a lot of evaluation criteria for potential use. The solution to that problem, it appears, is not to ignore evaluation of important dialogue manager properties but to intensify the work on early design support tools which can significantly reduce the amount of errors that would otherwise have to be identified, diagnosed and remedied as a result of evaluation.

Space does not permit presentation of ongoing work on filling the templates for the 37 dialogue manager evaluation criteria. Some observations regarding the list are that (i) the list of evaluation criteria provides a good illustration of the complexity of dialogue manager evaluation, reflecting the core "hidden" role of the dialogue manager in many SDIs. (ii) Few of the evaluation criteria in the list are straightforwardly quantitative, at least for the time being. (iii) Some of the criteria which are quantitative, cannot be applied to the SDIs performance as a whole but must be applied diagnostically, for instance by inspecting interaction log files to see whether, e.g., the dialogue manager adequately supports the performance of correct co-reference resolution or ellipsis processing. (iv) Many of the criteria are qualitative. (v) Some criteria, and not the least important ones, are subjective and must be measured through interviews, questionnaires and other contacts with the users to elicit those along with their subjective impressions from interacting with the system that may be attributed to the workings of the dialogue manager.

The reason for the existence of so many qualitative criteria on the list is the highly contextual nature of most dialogue managers. A good meta-communication strategy, or a good feedback strategy, for instance, may solve problems arising from the insufficiency of other elements of the dialogue manager or of elements in the speech or

language layers as well. If, for instance, the system lacks barge-in, the dialogue manager may have to be designed differently from its design for a similar SDI that does have barge-in.

The following example shows the filled template for the evaluation criterion on sub-task or topic identification success.

- **Sub-task or topic identification success**

A. What is being evaluated: the extent to which the dialogue manager succeeds in identifying the sub-task(s) or topic(s) addressed in the user's utterances.

B. System part evaluated and type of evaluation: module evaluation; qualitative. Quantitative evaluation is possible in SDIs in which task slots or translation templates are being filled directly from identified sub-task(s) or topic(s). In other SDIs, the dialogue manager may get the sub-task(s) or topic(s) right whilst still failing to get the user's contribution to the sub-task(s) or topic(s) right.

C. Method(s) of evaluation: design analysis, I/O tests. For high-complexity topic identification: Wizard of Oz data analysis, user-system interaction data analysis.

D. Needs and dependencies: virtually all SDIs need to do sub-task or topic identification, and this can be done in many different ways; sometimes linked to prediction success.

E. Life cycle phase(s): early design, implementation.

F. Importance of evaluation: high; a core measure of the successful working of a dialogue manager.

G. Difficulty and cost of evaluation: grows with the complexity of the task(s), user input complexity etc.

Given an SDI aspect, a checklist of possible evaluation criteria for that aspect represented as filled evaluation templates, and a particular application development project, the developer might proceed through the following iterations:

- in the *first iteration,* the developer selects the criteria which are relevant to the aspect-application pair, taking into account relevant constraints on the development process, such as cost;
- in the *second iteration,* the developer makes the selected criteria specific and applicable by making explicit the implicit conditions that apply to the development task at hand, such as which dialogue segmentation strategy the application will be using;
- in the *third iteration,* the developer plans when to apply the specified criteria during the development process, including

relevant parts of that information in the requirement specification;
- finally, in the *fourth iteration,* the criteria are applied in a methodologically sound manner as planned.

In the context of DISC, it is interesting to note that the above four-stage model can be used for 'meta-evaluation' of the development process. Central questions to ask during meta-evaluation are:

- did the developers select the right evaluation criteria given the constraints they had to observe?
- did they make these criteria sufficiently specific to their development task?
- did they apply the criteria correctly at all the development stages at which they should have been applied?
- what were the results?
- did the developers take adequate action in view of the results?

• Conclusion

This chapter has provided a brief introduction to the DISC dialogue engineering best practice methodology under development. The methodology includes a 'grid' that captures key properties of an SDI and its components, and a life cycle model that captures the development and evaluation process. We then focused on evaluation, describing the DISC conceptual apparatus. Drawing on the DISC concepts, an approach was presented for (i) how to systematically generate a comprehensive set of evaluation criteria for SDIs and their components, and (ii) how to thoroughly characterize evaluation criteria through the use of a common template describing the what, the when, the how, the importance, the difficulty, the cost, etc. of evaluation. The approach was illustrated by the case of dialogue manager evaluation. In brief, we have argued that if the SDI developer has access to a systematic inventory of the potential properties for evaluation together with best practice characterizations of the criteria for evaluating those properties, s/he will be positioned to efficiently evaluate any particular SDI or component, taking into account the actual constraints on the development process.

What has been presented is ongoing work. So far, only dialogue management has been addressed using the described approach. With respect to dialogue manager evaluation, it still remains to be seen exactly how detailed and informative it will be possible to make the filled evaluation templates and how much articulation and decision work is left for the developers, for instance with respect to details such

as how many users to involve in a certain test, how to collect and annotate data, etc. In any case, we believe that the basis for making those decisions becomes stronger with the DISC approach. Test with developers will help determine whether or not this is correct.

Our working hypothesis is that the evaluation template will turn out to be applicable not only to dialogue manager evaluation but to evaluation of all aspects of SDIs and their components.