Deliverable D3.6

Draft Proposal on Best Practice Methods and Procedures in Human Factors

April 1999

Esprit Long-Term Research Concerted Action No. 24823

Spoken Language Dialogue Systems and Components: Best practice in development and evaluation

DISC

TITLE	**D3.6 Draft Proposal on Best Practice Methods and Procedures in Human Factors**
PROJECT	DISC (Esprit Long-Term Research Concerted Action No. 24823)
EDITORS	Klaus Failenschmid (Vocalis), Laila Dybkjær (NIS)
AUTHORS	Klaus Failenschmid, David Williams (Vocalis) Laila Dybkjær, Niels Ole Bernsen (NIS)
ISSUE DATE	28.04.1999
DOCUMENT ID	Wp3d6
VERSION	2
STATUS	Restricted
NO OF PAGES	65
WP NUMBER	3
LOCATION	
KEYWORDS	WP3, human factors, best practice

Document Evolution

Version	Date	Status	Notes
1	02.99	Draft	For DISC partners' comments
2	04.99	Final	Final version of draft proposal

Abstract

This DISC report presents a draft proposal on best practice in human factors related work in the design and implementation of commercial and research spoken language dialogue systems. human factors cover all those aspects of interactive systems design which are related to the end-user's abilities (perceptual, cognitive and motor), experience (system specific, domain specific and common sense), goals (both interactional, i.e. related to maintaining the relationship between communicating parties, including ritualistic communication such as politeness, and transactional, i.e related to the external goal of a communication, e.g. finding directions to the theatre, and organisational/cultural context. The report is a further elaboration of DISC deliverable D1.6. D1.6 was on current practice in human factors and included examination of a number of existing spoken language dialogue systems in terms of a skeleton best practice framework for design process and design practice. The systems were Verbmobil, Waxholm, Vocalis Operetta and the Danish Dialogue System.

Contents

1. Introduction

What makes an artefact usable?

> A feeling of control, a good conceptual model, and knowledge of what is happening are all critical to ease of use. The controls must be recognizable, it must be easy to remember their function and operation, and they must provide immediate and continual feedback about the state of the system.
>
> […] Understanding comes about when the system presents a clear conceptual model of itself, making the possible actions apparent.
>
> (Norman, 1998, p. 178)

This report presents a draft proposal on best practice in human factors related work in the design and implementation of commercial and research spoken language dialogue systems (SLDS). SLDS can be separated into two broad categories. Firstly, there are the more 'traditional' systems which have grown from tone-based Interactive Voice Response applications. These have a dialogue structure (including conditional branches) which is defined at compile-time, i.e. a particular dialogue flow can be specified *a priori*. At the other extreme are *dynamic* dialogues where the dialogue is generated at run-time as a function of the recognised input, or more specifically, the information that has been provided by the user. Within this range of functionality, there are systems which have a mixture of both styles by changing aspects of the dialogue at run-time dependent upon static user data, e.g. usage profiles.

Since this paper is not concerned with the natural language processing (NLP) aspects of dynamic systems, we will confine our discussions to the post-NLP processing manifestation of the dialogue where human factors decisions are similar for both static and dynamic dialogue systems. Where this is not the case, the distinction between the two types of systems will be made clear.

Before defining human factors it is important to elucidate the range of types of communicative input into a system which can be regarded as spoken language dialogue systems. These are shown in the following list in order of increasing technical complexity. Example interactions are given. S means System; U means User; the words in bold typeface are recognised by the SLDS:

1. *Tone (DTMF[1]) Detect*: Tones produced by pressing keys on a telephone keypad.
   Interactive Voice Response (IVR) systems use this functionality and produce voice output.
   S: Press 1 for sales department, 2 for marketing,...
   U: [key 2 pressed on telephone keypad]
2. *Grunt Detect*: Presence of grunt noise or silence as inputs.
   S: ...or stay silent to speak to an operator.
   U: [Caller stays silent]

[1] DTMF: Dual Tone Multi Frequency. This term refers to the tones produced by pressing a key on the telephone keypad.

3. *Yes/No*: Only recognition of the words Yes or No.
   S: Do you want to speak to an operator, yes or no?
   U: **Yes**
4. *Isolated Word*: A small vocabulary of input words (e.g. command words).
   S: .. say balance to receive your current account balance, or operator to ...
   U: **balance**
5. *Word Spotting*: One input word can be recognised amongst non-valid words in an utterance.
   S: You can ask for your balance, ..., or speak to an operator.
   U: Can I speak to an **operator** please
6. *Multiple Word Spotting*: Multiple words or phrases which conform to super-lexical grammars can be recognised (e.g. connected digits).
   S: Please state your account number
   U: My account number is **3748 81847**
7. *Natural Language Processing*: Input utterances are parsed syntactically and semantically. More often in limited domains. Allows unrestricted input with co-references (e.g. anaphora, ellipses). Prosodic information in the utterance can be evaluated. Input can be spontaneous and continuous.
   S: How can I help?
   U: **Can I book a return ticket from London to Paris please**

- Additional features which can be used in the cases c-g above, are:
  - Confidence measures: give some measure of certainty that word X has been recognised correctly.
  - N-best: related to confidence. An ordered list of words with attached confidence measures.
  - Talkover (barge-in): users can interrupt the system by speaking (i.e. speak while the system is speaking). Recognition can be carried out on such utterances.

Human factors cover all aspects of interactive system design which are related to the end-user's abilities (perceptual, cognitive and motor), experience (system specific, domain specific and common sense), goals (both interactional[2] and transactional[3]) and organisational/cultural context (del Gado and Nielsen, 1996). Whilst the remit of this field is broad, theoretical and practical work tends to occupy a variety of small niches with few unifying approaches defining interactive system design on all of the dimensions noted.

The present report is a further elaboration of DISC deliverable D1.6 on current practice in human factors, which was heavily based on analysis of a number of existing spoken language dialogue systems and evaluation of current design practice against a skeleton best practice framework. The systems were evaluated from the point of view of human factors by researchers who had not been involved in their development. The systems were Vocalis Operetta - an automated call handling system [http://www.vocalis.com/pages/products/

[2] Related to maintaining the relationship between communicating parties. This includes ritualistic communication such as politeness.

[3] Related to the external goal of a communication, e.g. finding directions to the theatre.

operetta. htm]; Waxholm, a multimodal system which provides information on boat traffic in the Stockholm archipelago (Bertenstam et al. 1995); Verbmobil, a spoken language translation support system for German/English and German/Japanese [http://www.dfki.uni-sb.de/verbmobil]; and the Danish Dialogue System, a telephone-based system for reservation of Danish domestic flight tickets [Bernsen et al. 1998]. For more complete lists of existing SLDS, see (Bernsen et al. 1998; Gibbon et al. 1997).

The present report presents a draft best practice proposal on Human Factors issues and discusses options, pros and cons for each issue, when relevant (Section 2). The issues are those identified as forming the DISC grid for Human Factors. The report also addresses best practice for the Human Factors life-cycle (Sections 3 and 4) with particular emphasis on evaluation (Section 5). Section 6 concludes the report. Appendix 1 contains a checklist which summarises important points from the Human Factors grid as well as from the life cycle. The idea is that it may serve as a quick reference list for the designer. Appendix 2 presents a draft template for the evaluation of aspects of SLDSs.

2. Human Factors in SLDS Design

This section presents the draft best practice proposal for the DISC Human Factors grid in terms of a set of issues which should be considered when developing SLDSs or components. Whenever relevant, different options for an issue are discussed including pros and cons. The Human Factors issues considered all relate to spoken language dialogue systems and components. A set of current practice issues were established on the basis of analysis of a range of existing SLDSs (Williams et al., 1998). The current practice Human Factors issues for SLDSs are in this report extended and improved to constitute a draft proposal for Human Factors best practice. This has mainly been done on the basis of our own analyses and experience. A developers' test of the issues proposed still remains to be done. The presentation of the grid issues in the following are structured around the following five main topics: detection and correction of errors, dialogue initiative and design, user expectations and user background, user modelling, and how to address users.

2.1 Issue: Error Detection and Correction Addresses only a small fraction of the problem

Human machine dialogues, like human-human dialogues, will rarely proceed without any need for some kind of communication which is not directly related to the communication goal but concerns the communication itself. This is called meta-communication. A key example of meta-communication is the need to confirm that certain pieces of information have been correctly understood. These can be in the form of an explicit request, e.g. Did you say twenty-one?, or a more subtle repetition, e.g. twenty-one?[4]. In both cases the goal is to make sure that important information has been understood correctly as well as offer the possibility to detect communication errors early on. Detecting such errors is essential for the effective completion of the communication, as shown by the examination of human-human talk (for examples see Falzon, 1990).

However, the detection of an error is only the first step, the error still needs to be corrected. Correction is needed if information has not been transferred successfully. This can happen for a number of reasons, e.g.:

- One party does not hear the other.
- One party misunderstands and provides an unsuitable response.

Again, the error correction can take a variety of forms, ranging from the repetition of the statement in question through to a entirely different kind of dialogue structure. Examples of error correction are given by Cheepen and Monaghan (1997). Here, transcriptions that were made of call centre recordings were examined for error correction dialogues.

In human-human communication, the error detection and correction dialogues form a natural and seamless part of the talk and yet, if analysed fully, reveal a complex exchange of information. Figure 1 shows an example from Cheepen and Monaghan (1997).

[4] Known as an ellipse, where the question contains an answer.

It is the complexity of the turn-taking within such a dialogue which presents the biggest obstacle to its being modelled in human-machine dialogue, even assuming that recognition is perfect.

Options:

A variety of strategies are available to the dialogue designer depending on the recognition technology they have available. In addition, the designer must also be aware of domain-related aspects of the dialogue, e.g. how important is it that a piece of information is transferred correctly between human and machine. These two areas, technology and communication domain, interact in a way which determines the optimum strategy a designer should follow. The interaction becomes evident when one considers the use of confidence level recognition technology. This allows the recogniser output to include a confidence measure for each recognised word. In addition, there may be the facility to provide a list of vocabulary words which are ordered by their confidence.

Agent	**Caller**
..but you said something about you wanted to change the name Miss Ward.	
	Yeh, the name should now be Mrs Franton
Mrs.	
	M. Franton
Franton	
	F-R-A-N-T-O-N
T-O-N	
	yes

Figure 1. Example of error detection and correction in a human-human dialogue.

Case Study: The use of confidence measures

A small company requires an automated operator's assistant to answer incoming calls. A voice routing system is suggested which has a confidence level facility and provides the following dialogue:

***System**: Welcome to the Acme Consumables automatic switchboard. After the tone, please say the first and last name of the person you would like to speak to and I will connect you.*

***User**: David Williams*

***System**: Please hold while I transfer you* (High confidence)

A key requirement of the company is that calls are never routed to the Managing Director by mistake. For this reason, even on high confidence the system asks for confirmation of the name that was recognised.

***System**: Was that David Williams, yes or no?*

Case Study 1. Switchboard: The use of confidence measures.

Pros:

By using this facility, the dialogue flow can be diverted depending on the confidence level for a particular word (or phrase); a low confidence suggests that the system may not have recognised the spoken word correctly (error detection) and therefore a sub-dialogue is required to correct this error.

Cons:

However, the reliance that is placed on confidence measures must be tempered by the fact that confidence levels can mean that good words, as well as bad words, can be rejected. Given this danger, if a particular piece of information is safety critical, economically significant or would cause embarrassment if it were misunderstood it is necessary to add an explicit confirmation dialogue, even for high confidence results (for an example, see the Case Study 1).

2.2 Issue: Dialogue Initiative

There are a variety of ways that a spoken dialogue can be constructed.

Option: System Directed

The system leads the user, asking them questions which determine the dialogue flow. This type of dialogue will be most common in systems which must support novice users or which have a number of pre-defined steps which must be followed in order to complete a transaction or to collect information, e.g. an account number.

Option: User Directed

Giving the user control is more likely to support more experienced users. In this case, the system provides a range of functions which can be accessed from a main prompt such as 'Which service?'. A key issue here is how to make the user aware of the available functionality; this will be discussed in the next section.

System	User
Welcome to Acme Banking. Which service do you require?	
	Transfer
What type of account do you wish to transfer money from?	
	Banking
What type of account do you wish to transfer money to?	
	Savings
How many pounds?	
	Fifty
How many pence?	
	Zero

Figure 2. Mixed Dialogue Showing System Directed (bold) and User Directed (plain).

Option: Mixed Initiative

There may also be the case where the system directed and user directed modes are mixed in a single dialogue. For example, a telebanking dialogue may begin with a user directed service selection but later may provide a system directed dialogue to guide the user through fund transfer (See Figure 2).

2.2.1 Issue: Interaction mode

Dialogue initiative is strongly influenced by the mode in which the interaction between the user and the system takes place. The two main modes used in SLDSs are:

Option: Transactional Mode

The dialogue between the user and the system is focused towards achieving a common goal quickly. It is assumed that the user knows what type of information is needed and prepared to supply this information in very short utterances (e.g. isolated words). Initiative is usually assumed by the system and system output is constructed to support this. Command based systems with a menu structure (see discussions below) typically operate in a transactional mode.

Option: Interactional Mode

In an interactional mode initiative usually lies with the user and information between the user and the system is exchanged through longer utterances which can contain numerous pieces of information. The 'dialogue route' to solving the communicative problem is decided upon during the interaction. SLDSs that use a form filling approach (see discussions below) typically operate in an interactional mode.

2.2.2 Issue: Menu Design

An important aspect of dialogue design is making the user aware of the system's functionality and how it is accessed. In a visual interface this may be done in a variety of ways, see Figure 3. All of the represented methods make use of the permanent and parallel nature of the visual modality. Numerous menu options can be recognised in parallel and at any time. Unfortunately, speech offers transience and serialism[5], making the designer's job much harder. However, it also offers a departure from the rigid structuring of menus that do not allow the user to access commands in random order.

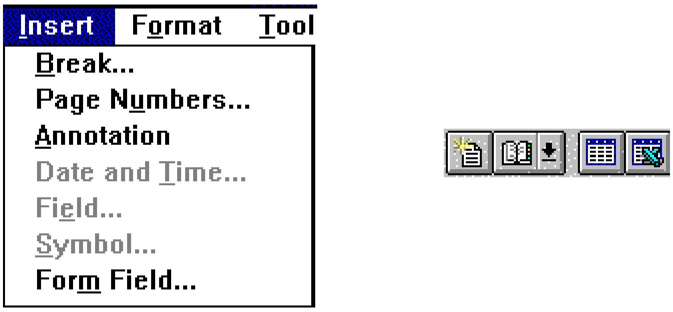


Figure 3. Visual Menus.

The advent of ubiquitous telephone network-based speech applications has increased the need to provide intuitive IVR (Interactive Voice Response) interfaces for the automated 'front-ends' allowing users to chose one from many services. Two related problems arise for the interface designer. How to design for users who have a wide range of usage experience and how to provide an intuitive interface for the increasing variety of network-based applications. Services such as call forwarding, call blocking, fax-mail, voice-mail, voice-dialling are now commonly available to subscribers and public users.

Traditionally, automated IVR for multi-function systems have been based on linear and hierarchical menu structures since these are well known in graphical/textual interfaces. However, other approaches have been attempted, including 'skip and scan' (Resnick, 1992).

Option: Skip and scan

The 'skip and scan' style of menu navigation (Resnick and Virzi, 1992) gives users more control over system output. By using a cassette-player analogy users can move backwards and forwards through menemes of a particular menu by using 'next' and 'back' Once the appropriate meneme has been found, it can be selected by saying 'select'. This process is repeated for further submenus.

Pros:

The advantage of this method is that users do not need to hear all of the menemes in a menu allowing them to quickly select their required option once they hear it. In addition, since

[5] It is possible to provide parallelism in spoken dialogues using non-verbal sounds (see Section 0) or stereo positioning.

navigation is reduced to generic commands, there is no need to explicitly include these commands in the prompts. For example, a dialogue can be reduced to:

System: 'File Menu: Include'

User: 'Next'

System: 'Save as Text'

User: 'Next'

System: 'Log Message'

User: 'Select'

The method can be further enhanced by using talk-over allowing the shortened prompts to be interrupted.

Cons:

Performance time does not seem to differ much from several other methods. Comparisons between different methods (Jack et al., 1996) have focused on search time and found little difference between 'shallow and wide' (few levels, numerous commands per level), 'deep and narrow' (numerous levels, few commands per level) and 'skip and scan' methods.

The structure of a menu in a command based SLDS has a great effect on the usability and naturalness of the system and different approaches to menu design will now be discussed. An approach, namely form filling, for non-command based systems will also be introduced briefly.

Option: A linear main menu or help prompt listing all of the functions (ordered sensibly)

Pros:

This makes all of the system functionality evident and, if talkover[6] is available, can cater for both novice users (who do not know what commands are available) and expert users (who know the commands available and can interrupt the prompt).

Cons:

However, some disadvantages are inherent:

- For the novice user, the long list of possible commands places too much of a strain on short-term memory.
- If no talkover is provided, multi-function systems are very slow to use since the caller must hear all of the command names before being able to control the dialogue. If an unfamiliar command is required, placing the list in a help prompt causes a similar problem.

As with visual menus, it is important that mememes are ordered categorically, e.g. editing commands together, navigation commands together.

Option: A Menu-hierarchy

The linear (flat) function list is grouped into menus and sub-menus so that the number of available command words is limited. Speaking a valid command word either executes the associated function or makes a sub-menu available with an other group of command words.

Pros:

[6] A facility which allows callers to interrupt the system output.

This approach allows the semantic grouping of functions making memorising group members easier. It may also support random access to any level via command names.

Cons:

However, the following disadvantages are identified:

- If the hierarchy becomes particularly deep, i.e. many sub-menu levels, then additional commands must be provided for navigation. This compounds the multi-function problem since the user must learn additional non-application commands, e.g. "back", "next", "where am I".
- The experienced user will need to navigate through the sub-menus to reach the function they require.
- Users are required to mentally 'visualise' the menu hierarchy in order to effectively navigate it. This can be problematic for a complex structure often leading to the user becoming ‘lost’ in the menu-space.

Option: Pseudo Sub-Menus (PSMs)

The advantage of sub-menus, without the drawbacks for experienced users, can be obtained by providing a *pseudo* hierarchy using PSMs. In this approach, sub-menus do not provide a confined command space as with the normal hierarchical approach; a space which must be navigated in order to get to the lower function ‘leafs’. Instead, PSMs can be considered as extra-help prompts which group the many available functions and give the function names and sit at the same level as commands.

Thus, from the top-level prompt all of the functions can be accessed by their name. As well as these, each of the pseudo-menu names are active. Selecting these gives a prompt which simulates a sub-menu listing each of the functionally related menemes. However, all of the functions are still available from this prompt; thus giving a flat structure.

Pros:

The advantages of this approach are mainly related to the experience of the user. The inexperienced user will not be presented with a long list of functions but functions grouped into intuitive sub-menus. However, once the user becomes more experienced with the available commands, they can access all of the system functions directly from the ‘Which Service?’ prompt rather than having to negotiate the hierarchy of ‘concrete’ sub-menus. Interactions like in a linear list as well as like in a menu-hierarchy are possible.

Option: Form filling

The PSM approach in command based SLDSs is a step towards a dialogue management approach that is based on form filling strategies. The core of this strategy is the provision of a form which contains slots for required information for a particular task to be achieved when interacting with the SLDS. For a boat time table such a form could hold information about the place of departure, the destination and the time of departure. The dialogue manager then tries to route the dialogue such that all the slots are filled in the interaction. Depending on what information is already in the form the dialogue manager queries missing information. The users are therefore not constrained in the order in which they give information to the system. The order and the number of distinct utterances in which the information is given can be chosen freely by the user.

2.2.3 Issue: System output design

It is somewhat artificial to separate system output wording from dialogue flow but it is suitable for the current discussion. The user-facing part of a spoken dialogue is the prompting, i.e. the system's voice[7] and it is here that the designer provides a representation of the system state. System output utterances (prompts) are analogous to the visual components of visual interfaces, i.e. the perceptual properties (size, colour, contrast, shape).

Table 1 shows system output defined on four dimensions along with possible counterparts in visual interfaces.

Attribute	**Visual Counterpart**
Wording (e.g. politeness)	Organising metaphor, icons resemblance, text
Register (e.g. speed, pitch)	Type setting, colour, size, shape
Tone (prosody)	No equivalent
Real/Synthetic	Digitised images/geometric or drawn images

Table 1. Prompt Dimensions.

All of these aspects must be considered in the particular context of use within a dialogue flow. It is interesting to note that spoken words provide a rich variety of dimensions that can encode information, some of which have no parallel in visual interfaces. The main difficulty in the effective design of prompts is their evanescence along with the lack of specificity of spoken language and the requirement for highly contextualised interpretation. Given these constraints it is essential that the following areas are addressed:

- *Language*
  Before embarking on the design of a prompt list care must be taken that the vocabulary and grammatical structure matches the abilities of the end-users. For example, any domain specific jargon should be avoided where possible. It is important to point out that the wording of the system output affects the recogniser vocabulary. Key words used in system outputs have to be present in the recogniser vocabulary since users are likely to repeat them.
- *Prompt Context*
  It is essential that the designer is aware of what has occurred earlier in the dialogue. This will determine what words and register should be used, i.e. the context of the prompt. For example, as a user moves 'deeper' into the dialogue it may be justifiable to shorten prompts. There may also be particular pieces of information that need repeating.
- *Communication Context*
  It may be necessary to consider the global context of the interaction. For example, a

[7] The system voice can be augmented by non verbal sounds or a variety of speakers (see James, 1997).

banking dialogue will have its own ‘etiquette’ or speech style. Work by Williams et al. (1998) suggests that such highly transactional domains require a particularly curt style of speech which does not contain politeness words, personal pronouns or anthropomorphism.

2.2.4 Issue: Type of system output

In addition, the type of system output needs to be considered since this has a strong effect on the kind of response a user is likely to give. There are three main types of system output:

Option: Complete Speech Recordings

Whole messages are recorded by a professional speaker and played to the user as a whole. If the prompts are carefully recorded, this is the highest quality speech output that can be used.

Option: Concatenated Speech Recordings

With this method only recordings of parts of phrases are made by a professional speaker. When a complete phrase is to be played out to the user, relevant parts are concatenated together and then played to the user as one utterance. Recording such prompts is more complex than recording individual phrases, since intonation of items changes according to the position within a phrase. For instance, intonation for individual digits varies according to the position in a string of digits. Concatenated speech is regularly used to speak back telephone numbers to callers, however. Only one recording per item (e.g. digit) is normally used which results in ‘robot-like’ utterances.

Option: Synthesised Speech

The most versatile means of output generation is the use of a Text-to-Speech (TTS) system. Such a system is capable of producing synthetic speech output from a text string. Although this allows the production of virtually any prompt, it has limitations and disadvantages. The speech quality is usually much worse than pre-recorded speech, and it can be very difficult to use a Text-to-Speech system to correctly pronounce certain classes of words such as names and place names. Current Text-to-Speech systems either produce prosodically rich speech output and low comprehension, or high comprehension and low prosody.

Pros and cons:

In contrast to concatenated speech which can sound 'robot-like' and synthesised speech which does not (yet) imitate real speech very well, complete speech recordings can be easily perceived as 'life-speech' from a real person. Users are therefore more likely to respond in a conversational manner (e.g. using long utterances) to a system that makes use of complete speech recordings. Speech output that is more obviously created by a machine (concatenated prompts or synthesised speech) attracts more constrained responses. Hence, not only the degree of naturalness in the system responses, but also the capabilities of the speech recognition technology that recognises and interprets the user's utterance need to be considered when deciding on the method of speech output.

2.2.5 Issue: Cues for Turn-taking

System output also must encourage users to speak at the appropriate time and be passive when no speech can be processed. Thus at the lexical level, a typical human-computer dialogue in a spoken language system consists of two stages, system output and user input. Critical to these stages are the cues that allow effective turn-taking to be carried out, i.e. cues for the user to speak (and the system to 'listen') and vice-versa. Even in systems that do not follow a rigid structure of turn-taking it is important to clearly signal when input from users can not be accepted.

Cues may be explicit, e.g. 'Please speak after the tone <beep>' or implicit, e.g. the user stops talking. As with human-human conversation, a good proportion of turn taking clues are given by lapses in talk, i.e. silences. These potentially ambiguous representations are resolved by a variety of means such as contextual semantic, syntactic and prosodic information or physical gestures, e.g. eye-brow raising.

However, systems that use speech-only as input medium cannot respond to non-verbal cues. Furthermore, in automated spoken dialogues, silences on the system's part may not be so easily resolved if they are of the implicit kind. In addition there is the requirement for basic signalling of an open channel, i.e. the system has not crashed. What is lacking in current (speech-only) systems is any explicit representation of these system states (the Waxholm system has an animated face on a monitor which informs the user of the state the system is in).

Whilst the meaning of the initial silence may be implied by the preceding prompt, e.g. questioning intonation, the move from speech processing to application processing is rarely explicitly represented in the dialogue. A consequence of this may be, for example, that if the user misinterprets the silence after finishing speaking to mean that the system has not recognised their speech, they may speak again. This can have two consequences depending upon whether the system is processing the speech or is in the application state (e.g. accessing a database). In the former case, the user's speech will be processed as part of the original utterance and could well cause a misrecognition. In the second case, the users utterance will be ignored, causing an unexpected system response if the utterance was not simply a repetition. This sort of misunderstanding is often only detected in a later state of the dialogue.

The problem lies with the paucity of representation for a number of system states. Since users rely on gaining application knowledge from the interface representation of the system state, without this knowledge the user is liable to act erroneously or react negatively to unexpected system output. What is required is a solution based on adequate and intuitive representations.

Sound	**Referent**
Stylised wind chimes	Top Level prompt, e.g. Which service?
Final resolving piano chords	Valediction
Several taps on hollow bottle	Name management mode
Two discordant chords	Error

Table 2. Non-Verbal Auditory Mappings (from Dutton et al. 1997).

This has been addressed by a number of authors (Brewster et al., 1994; Dutton et al., 1997) who suggest that non-verbal sounds in a dialogue would provide additional information on system state. For example, error conditions (misrecognised or invalid inputs), changes in dialogue mode (vocabulary, grammar) or greeting and valedictory messages. Possible mappings are shown in Table 2.

2.2.6 Issue: Help Design why under initiative? No pros and cons

If a truly user-centred design process is to be followed the designer must initially identify the target end-user population. This analysis will allow the level of help required in the dialogue to be decided. All automated dialogues must be designed to support the eventual end-users of the system - the callers - and consideration must be given to two major aspects of the caller/system communication. First, the caller must be able to understand what the system is capable of, and second, the caller must understand how to proceed with the dialogue in order to achieve the goal(s).

Help information may be specific to the current context or provide generic advice on the available services or how to speak more clearly. The information could do the following:

- Tell the user what has gone wrong (in the user’s not the system’s language).
- Tell the user how to get out of the help message.
- Encourage the user not to get into the same situation again.

Help messages should not be as short as possible. Often a certain degree of verbosity is required to explain the problem (and ways to resolve it) to the user. If an error loop is encountered the system output for each error loop should be adjusted following the guidelines above.

Providing useful help mechanism remains a difficult task not only because the kind of help needed is highly user specific, but also because the task domain impacts upon it.

Help information is intended both as an addition to and a respite from the mainstream dialogue and can be portrayed in a variety of ways:

Option:

As an implicit part of the dialogue, e.g. command names made explicit or example utterances provided ("Say connect to be connected or operator to speak to an operator").

Option:

As an alternative part of the dialogue which must be explicitly requested by a ‘help’ command. This includes:

- Demonstration utterances.
- A more in depth description of commands.
- A sample dialogue.

Option:

Automatically enabled if the user is having repeated misrecognitions (detected by the system through confidence measures or explicit disconfirmations). This includes:

- Progressive or incremental help (each time a user encounters recognition problems, new more explicit help is provided; from list of command words to example utterances)[8]
- Suggestive prompting (those alternative commands which have not been used are suggested).

2.2.7 Issue: Dealing with Naive, Novice and Expert Users

Different application domains (e.g. banking, ticket reservation) may share some dialogic features, such as menu navigation commands, which will allow callers to use experience gained interacting with one system in the interaction with an other. Additionally, the designer must bear in mind the need to cater for the absolute novice - i.e. the caller who has never before used an automated system. In other words, callers do not bring the same initial kinds of knowledge, experience and abilities to different application domains. Key areas where differences in caller characteristics may arise are:

- Familiarity with the general domain
  - e.g. in a telebanking domain, does the caller know about statements, balances etc.?
- Familiarity with automated spoken dialogue systems
  - does the caller know roughly what to expect when talking to a machine?
- Caller status
  - is the caller either a member of the general public who may use the system only rarely, or a (potential) expert who may (in time) use the system with great frequency?
- Caller motivation
  - what is the advantage to the caller of using an automatic system?
  - is the caller paying for the call?

Given these different knowledge types users can be broadly categorised into:

- *naive users*
  no experience of domain or spoken dialogue systems
- *novice*
  experience of domain and/or spoken dialogues. Some experience with specific application
- *expert*
  frequent users of the application and expert in the domain

A given dialogue does not need to support all of these users, rather the designer should be aware of what the typical user will be. Of course, there will always be naive callers to a system, the key is how often they use the system and therefore how quickly they will become experts. Ideally, a dialogue should provide for this transition if a system is to be used frequently. If this is not the case, help information should be ‘near the surface’ of the dialogue since there will be no expert users to be hindered.

[8] This facility is used in the Vocalis VAD (SPEECHtel system)

Case Study: **Serving different levels of experience**

A voicemail system is required which is driven by voice and will allow users to communicate with other account holders. To provide a more personalised service the following user profile is defined:

Personalised Greetings

Login Count

Usage Count per Function

This log is used to allow prompts to be shortened for an expert caller (called in many times over a short period)

What do you want to do? -> What now?

You have received a new message -> New message

Case Study 2. Voicemail: Serving different levels of experience.

Examples of dialogue styles which allow the transition from novice to expert are:

- Dialogues using *talkover* or *barge-in*. The long prompts required for new users can be interrupted by experts.
- A progressive help mechanism that begins with a prompt for an expert, e.g. 'Which service?' , but quickly lengthens if a caller is having problems, e.g. 'The services available are...'
- A change in prompt style which is more suited for a particular caller (see Case Study 2 above).
- Suggestive help mechanisms which reveal increasingly more complex aspects of the system's functionality to experienced callers.
- Provide a backup human operator for calls where the user experiences persistent difficulties.

The ideal situation can occur in systems that have some form of user identification. This allows a specific user's usage pattern to be matched to the relevant help strategy (see User Modelling and Case Study 2).

2.3 Dealing with User Expectations

Two of the main reasons for users to interact with an SLDS are to gather information (e.g. boat timetables), or to instruct an automated system to perform a certain task (e.g. call routing). Therefore, a large number of interactions between a dialogue system and a user tend to be predominantly task-driven – the idea is to use the dialogue system as a tool to perform a task. To support the user in successfully achieving such a task, an SLDS should facilitate task-driven interaction that takes into account a user's expectation of the functionality and capabilities of the system.

Therefore, the core function of the dialogue interface consists of matching a user's expectation of (and experience with) the task and the representation of this task in the SLDS. This is shown in Figure 4. The user interacts with the system with a certain expectation about the functionality of the system in mind. This is influential in shaping the behaviour of the user. In order to support a 'natural' interaction there is a need for closely matching these user expectations and the resulting behaviour with the system-internal representation of the task and domain. The system-internal representation of the task makes use of expectations about the target users' behaviour (interaction model) as well as information about the functionality that the system supports.

Usability problems become apparent when the user's *model of the system* (its perceived functionality and capabilities) differs from the *actual system* (actual functionality and capabilities). It is the role of the dialogue interface to make sure that such a mismatch does not take place.

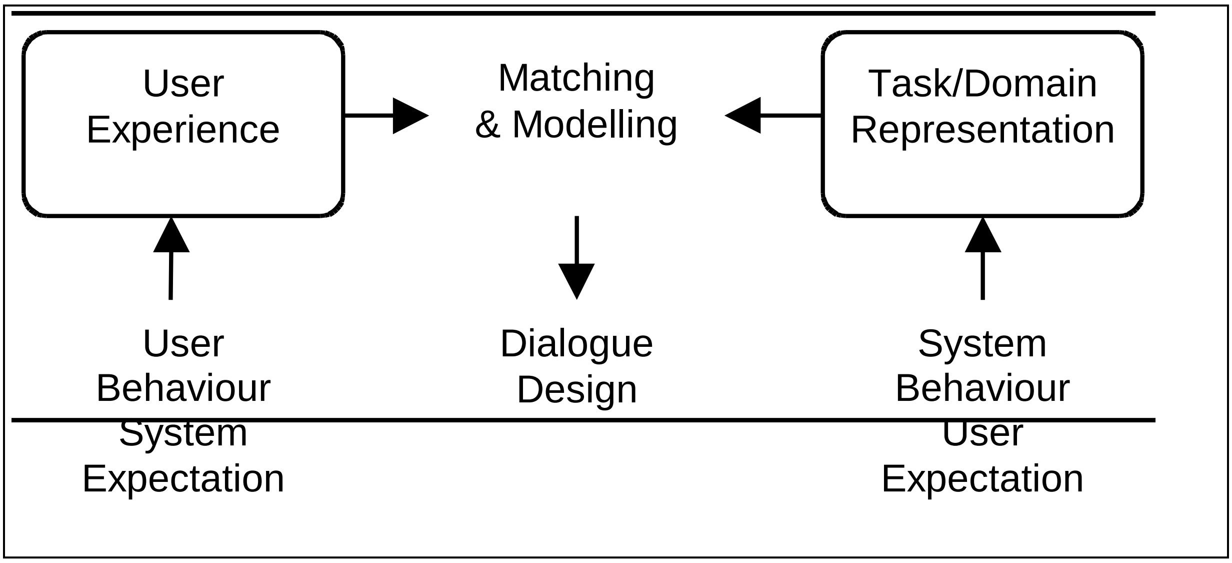


Figure 4: The role of dialogue design.

Designing a dialogue interface that prevents a mismatch of perceived and actual capabilities of a SLDS is extremely complex and it might therefore not be desirable (or possible) to design 'a perfect interface'. Rather, it might be more practical to be aware of the type of problems that can lead to a mismatch than provide error recovery strategies that offer a way out of a 'mismatch situation'. There are a number of issues that need to be taken into account when designing a dialogue interface that is sensitive to a user's expectation and consequently reduces the possibility of a mismatch between perceived and actual capability are the following. These issues are:

2.3.1 Issue: User experience

Users may bring with them knowledge from other systems, which they think is relevant to the new system. For example, they may be used to a system where word spotting is employed so that key words can be spotted amongst noise words, e.g. "I'd like to place an *order* please".

The user then moves to a system that only recognises isolated words, e.g. "*order*". Serious problems can arise when the second system does not provide out-of-vocabulary rejection, but rather tries to match the long phrase to one of the isolated words allowed in the vocabulary. A misrecognition is almost certainly the result.

A similar phenomenon can occur with respect to expected phrases[9] in the input utterance. This incorrect analogical reasoning can cause 'unexpected' recognitions by the system and degrade overall performance (Figure 5) (for more examples see (Basson et al., 1996).

…

[S]: I am sorry, there is no answer. Do you want to leave a message for Paul Smith?

[U]: No problem.

[S]: Please say the name of the person you want.

…

The user's utterance ([U]) is recognised as 'no' because wordspotting techniques are used. However, the user had intended a positive confirmation to leave a voicemail message.

Figure 5. User expectation – perceived and actual grammar coverage.

2.3.2 Issue: User adaptation to system output

Current practice analysis, as well as research has shown that as users spend more time with a system they adapt their verbal behaviour both to the system's apparent capabilities and to the style of language the system uses (Franzke et al. 1993, Zoltan-Ford 1991, Gustafson et al. 1997). An equivalent process is demonstrated in human-human dialogue where the participants settle on an 'operational' language that they are both able to understand (e.g. 'motherese', a receiver-adapted style of communicating between mother and infant). These may contain a restricted vocabulary, grammar and semantics.

Franzke et al. (1993) showed that subjects interacting with a speech system over a human operator used fewer words and less complex grammars. Zoltan-Ford (1991) and Gustafson et al. (1997) showed that user input is significantly shaped by system output. In Figure 6 parts of the formulation used in the system announcement is re-used by the user in the response. It is crucial to consider such user behaviour when designing an SLDS since it can have a severe impact on the performance of the final system.

If, for instance, real-life speech data is collected using a system announcement that prompts the user to only say the departure city (e.g. '[S]: Say your city of departure; [U]: Stockholm'), recognition performance in the final system that makes use of a different system announcement (e.g. '[S]: Where do you want to depart from?; [U]: from Stockholm') might be quite poor. This is because of the tendency of users to adapt their verbal responses to the system output wording.

[9] A speech recogniser is normally constrained by grammars which define the type of phrases that can be recognised. The grammars can be very precise and therefore limit the variety of input phrases that can be recognised / understood by the system. For a more detailed discussion of input grammars refer to D3.2 and D3.4.

This means that care must be taken not to encourage callers to use language that the system does not understand.

…

[S]: Var ifrån vill du åka? *(Where do you want to travel from?)*

[U]: från Stockholm. *(From Stockholm.)*

…

The user repeats part of the formulation used in the system question ('from…').

Figure 6. User adaptation to system output [taken from (Gustafson et al. 1997)].

2.3.3 Issue: User's understanding of the system functionality (Introductions)

A successful interaction between a user and the dialogue system can only be achieved when the user has a valid understanding of the functionality of the system. It is therefore important to 'educate' the user with respect to the kind of things that can be achieved with the dialogue system and the way in which this can be done.

Options:

For instance, for a dialogue system that provides a service for a well-specified group of users which can be targeted directly (e.g. users of an automated telephone banking service), the target users can be informed about the functionality and capabilities of the automated service by mailouts that also introduce the service. This printed information can also contain example phrases a user can use when interacting with the system.

Pros:

The advantage of providing written information about the service functionality and example phrases/interactions is that users can learn about the system 'off-line' while they are not interacting with it. It is also possible for users to acquire a comprehensive view of the system which they can refer to whenever they need to. They can start to understand the limitations of the system. Successful interaction patterns can be learnt by following interaction examples that are given in the printed materials.

Cons:

Only providing written information about the service is very dangerous because there will be a substantial number of users who will not read this information. Furthermore, an advanced service might offer a substantial amount of functionality which, in turn, requires lengthy and complex descriptions. Although having read all the information, a user might not be able to recall the information needed at a particular part in the dialogue with the system.

Also, some users might try and read the instructions or information while they interact with the system. The likelihood that system announcements are not heard and understood completely rises and unexpected answers to system questions can be the result.

If printed material is kept very specific with regards to the way in which the user should interact with the system, changes to the dialogue system can result in serious problems since they run the risk of not matching the detailed information distributed to the user. The information held by the user at the point of changing the system needs to be taken into account when making substantial changes to the dialogue interface.

Options:

Quick reference cards with a very short description of the vocabulary of the dialogue system as well as a visual representation of the underlying menu structure can be provided to users for relatively complex systems that use a menu structure. It is important to provide quick reference cards in a (physical) format that can easily be used in the context in which the user is likely to interact with the dialogue system. For instance, if the service is mainly used from a workplace (e.g. desk), a small single-sided piece of paper is more useful than double-sided reference card that can be folded up and put into a wallet. While the former type can easily be attached to a noticeboard above the desk and be read 'glancing' at it, the latter type is more appropriate for use in varying locations.

Pro:

Quick reference cards are a convenient way of representing a menu structure in a visual way. Also, information about the active vocabulary (keywords) can easily be provided. Instead of listening to potentially long 'help' or 'how-to-use' messages by the system users can quickly glance at a textual (or graphical) representation of the same information.

Con:

A quick reference card should never be the prime source of information for a user. A substantial number of users might not have access to the reference card when using the system and therefore require assistance from the system.

A major consideration about providing quick reference cards with representations of the system's menu structure is that changing the menu structure (and/or the vocabulary) can introduce serious problems since the printed version of the system doesn't match the actual system. Introducing changes into the dialogue interface will therefore be much more difficult and prone to errors.

2.3.4 Issue: User's understanding of the structure of the interaction

Dialogue systems that are used by the general public by a substantial number of novice or casual users can provide information about how to successfully interact with the system in an announcement at the beginning of the dialogue. Instructions such as "Please always answer after the tone" as well as short example phrases ("You can say, transfer twenty pounds from my current account to my savings account") can be given. Care must be taken that all introduced interaction patterns are used consistently throughout the whole dialogue. This is because the patterns that are introduced explicitly, as well as the ones that are introduced implicitly (e.g. sound of tone that signals a user's turn) lead to a set of expectations by the user. Important interaction patterns that shape a user's expectation of the interaction with the dialogue systems include:

- the grammatical structures used in systems announcements
  e.g. passive voice or active voice
- the instruction to the caller about when to answer
  e.g. "after the tone, please clearly say..."
- the frequency of the tone used to signal that it is the user's turn to talk
- the length of time which occurs after the end of the system announcement and the beginning of the tone that signals a user's turn

Pros:

Explaining interaction patterns and keeping them constant throughout the user-system interaction enables the user to predict the next stage in the interaction. For instance, if the pause between the end of a system announcement and the tone that signals a user's turn is kept constant throughout the whole application it is possible for user to know when a response is likely to result in a successful (i.e. expected) response from the system.

Cons:

None.

2.4 User Modelling and User-Dependent Dialogues

2.4.1 Issue: Anonymity vs. Identity

Two types of speech systems can be defined, vis-à-vis their knowledge about the user and their activities. The first type is typical of utility applications for the general public, e.g. automated call centres, electricity or water metering. Here the user provides no identification, they can be anybody. In the second type of system, the users are given a personal account number which identifies them to the system. Such systems have a database record for each user which can contain a variety of dialogue, or more usually, domain-related information.

In addition to information gathered across whole transactions, there may also be the facility for information garnered between individual dialogue steps to be stored and used.

It is important to note that these two major types of applications do not only differ with respect to the user identification, but also in the core nature of the task. While the former is a type of customer service application that provides information to the user, the latter is a 'self-service' application that is used like a personal assistant. Therefore, a 'closer relationship' can be formed between the latter type of service and the user and there is a potential for learning interactional patterns.

The discussions in this section mainly address 'self-service' applications with users who reveal their identity prior to interacting with the service. Despite this, some aspects of the discussions are applicable to 'customer service' applications with a totally unknown user population. Where this is the case it will be pointed out.

2.4.1.1 Issue: Keeping a user profile

Applications that demand that users identify themselves to the system prior to using it can keep a user profile. It is common to store user data that is task related such as in an automated banking service. Here, customer specific information such as their address, the kinds of accounts they have, etc. is stored. This information is required in tasks supported by such a service. However, this task-related information can be supplemented by 'interactional' information. Such 'interactional' information can then be used to shape the dialogue flow. Case Study 3 shows an example of a 'self-service' application that makes use of a very limited user-dependent data record with 'interactional' information. User of this system will identify themselves through an individual access number which then allows the system to find the user specific information.

Case Study: User modelling

A voicemail system is required which is driven by voice and will allow users to communicate with other account holders. To provide a more personalised service the following user profile is defined:

Personalised Greetings

Login Count

Usage Count per Function

This log is used to:

provide recommendations of which services have not been used very often;

e.g. *I'd recommend you say play, back, delete*.

shorten prompts when usage exceeds a threshold for any given command

e.g. *What do you want to do?* becomes *What Now?*

Case Study 3. Voicemail: User modelling.

Options:

There is a multitude of interactional information that can be stored for each user. Examples of such information are:

- Date and time of last interaction with the system.
- Number of interactions with the system.
- Number of times specific functions have been used.
- Most frequently used functions.
- Preferred vocabulary (out of a list of synonyms).
- Number of dialogue steps to achieve a task.

Most of this information can also be collected in customer service applications that do not require users to identify themselves. However, the collection of the information is then limited to an individual session or across multiple users. The data should then only be used to inform further development of the dialogue interface (see also: (Failenschmid and Chase, 1999).

Pros:

The collection of interactional information provides means to shape an interaction such that it supports a user's style and way of interaction better. This information can also be used for the analysis of users' behaviour when interacting with the system. Such analysis can inform the further development of a dialogue interface (Failenschmid and Chase, 1999).

Cons:

The collection of extra interactional data requires the ability to store more user-specific data. Therefore, system resources need to be sufficient to collect this extra amount of data for the projected number of users of the dialogue system.

Data collected can be misleading when different users share one 'user-account' since then information is recorded with the assumption of tracking only one user's profile, although more than one user 'supplies' the data.

Options:

An alternative way of collecting user specific data for adapting the dialogue to users' individual requirements is to explicitly ask users for their preferences. This can either be done by providing an extra part in the dialogue that can be used to set individual preferences by, for instance, answering simple questions or by providing alternative means to do this such as visual web-interfaces to the service.

Preferences can also take on the nature of a user-specific vocabulary such as a list of names in a voice activated dialling service.

The specification of a user-specific vocabulary can also be done at set-up time. For instance, if an automated banking service is offered, user-specific account names can be supplied when applying for an access code to the system. This is often done by filling in an online- or paper application form.

Pros:

The user has the chance to shape the dialogue interface directly. Changes can be made early on without the need to interact with the system over a certain period of time before changes become apparent. Complex changes can be made using a graphical interface which is better suited to this sort of task.

Cons:

The effects of setting each preference must be clearly understood by the user, otherwise there is a risk that the resulting dialogue changes do not match the user's requirements very well. Furthermore, there is a danger that a whole new set of 'usability' problems is introduced by allowing users to manipulate the dialogue.

Offering users to set preferences requires that they not only learn how to interact with the system but also what they can 'manipulate'. This can be a complex task and might therefore not be beneficial to the user.

Providing the facility of adding a user-specific vocabulary must be coupled with the ability to review this vocabulary. This is because it is likely that users forget what entries they have in their personal list[10].

There is also a danger that users spend more time on learning about how to set their preferences (configure) for a particular service than it would take them to actually learn, or get used to, the interaction supported by the standard dialogue interface.

Options:

Tracking user interaction and then asking whether the interactional style should be changed.

Pros:

The user is in full control of the introduction of different preferences.

Cons:

[10] This is a phenomena known from research into other types of user interfaces (e.g. the ability to create aliases (links) under UNIX). For more information on this with reference to graphical and textual interfaces, see (Nielsen, 1993).

There is a need for explanation of what certain preferences mean and how they will affect the interaction with the dialogue system.

2.4.2 Uses of User-specific Information

The use of user-specific information in a 'self service' application will often depend on the needs of the application. However, a number of more generic uses can be identified:

Options: Prompt Adaptation

As users become more experienced with a system there may be less need to use verbose prompts. Knowledge about the user’s transaction history allows new shorter prompts to be used once the transaction count exceeds a certain number.

Pros:

Since users are aware of the functionality and capabilities of the dialogue interface, they only need short prompting to know what is expected of them in a particular part of a dialogue. 'Talkover' ('barge-in') allows users to interrupt system announcements and thereby speed up the interaction. However, it is known that a number of users do not interrupt the system and are therefore not able to interact more quickly with the system as they become more experienced. Reducing verbosity (i.e. length) in system announcements has a similar effect to interrupting the system in that it speeds up the interaction without requiring the users to change the way in which they take their turn (e.g. from speaking after the end of the announcement to interrupting the announcement.)

Cons:

Prompt adaptation has to be introduced carefully so that a consistent style in the dialogue is maintained. Also, changes in prompt wording need to be made in such a way that they do not encourage users to give slightly different feedback (Figure). Provisions should be made for users who return to a frequently used service after a relatively long time of not interacting with it. It is possible that in such circumstances users need more explicit prompting again. Adaptation of prompts should be introduced slowly in small and distinct steps so that it is almost transparent to the user.

Options: Dialogue Flow Adaptation

As well as prompt adaptation there is also scope for the dialogue itself to change. For example, more experienced users may wish to input multiple pieces of information at once rather than use a system directed dialogue. In such a case more complex (and possibly less robust) technology can be made available to the user by simply activating new ways of interacting with the system. In order to make users aware of this increased functionality 'suggestive prompts' (see below) can be used.

Pros:

More efficient ways of interacting with the dialogue interface can be supported by dynamically adapting the dialogue flow based on a user's interaction history. Users get the chance to interact with a system that supports them better in the particular way in which they wish to achieve a certain task.

Cons:

Obstacles can be introduced by not supporting previous styles of interaction. For instance, if the dialogue flow is changed such that it allows users to specify their request in a more complex way (more than one information token in a phrase), the previous style of interaction (e.g. only one information token per utterance) should continue to be supported with the same system response.
It is advisable to only minimally change the dialogue in order to avoid confusion on behalf of the user.

Options: More complex recognition

The use of anaphora in human speech is common. Such references can only be resolved if previous information is stored as the dialogue progresses. Given this is the case, the possibility of correctly interpreting 'Move the money to that account' becomes a reality. In truth, such complexity is not in the realm of commercial applications but will be soon as user populations become more comfortable with speaking to machines.

Pros:

A more natural way of interacting with the dialogue interface that does not constrain the user because of limitations with the technology used will be the result of the introduction of more complex recognition.

Cons:

However, there is limited information available with respect to user behaviour with such a system. The usability of such an interface in a real-life situation with real users needs to be analysed more completely before recommendations can be made.

Options: Suggestive Dialogues

Common user behaviour can be extrapolated from user data allowing systems to suggest particular dialogue strategies. This can take the form of 'suggestive prompting'. If the system can detect that a user carries out a task in a way that seems to be more complex than it needs to be (e.g. multiple turns to achieve a task when this could be done in one turn), a 'suggestive prompt' can be played out that suggests to the user how to carry out the task more quickly (Figure 7).

Pros:

System functionality that would normally be 'hidden' from users can be brought to their attention. Users can benefit from faster interaction. Since the core dialogue does not change users can opt to ignore the suggestion and continue interacting with the dialogue interface in a way they are familiar with.

Cons:

Extensive use of 'suggestive prompting' can alienate users because the dialogue system tells users how to interact. They might not feel that they are in full control, but rather need to be told how to do something properly, this can be perceived as being patronising. The wording of the 'suggestive prompts' should therefore be chosen very carefully.

…

[S]: Do you want to transfer money, ask for a balance or pay a bill?

[U]: Transfer money

[S]: Transfer from which account?

[U]: From my current account

[S]: Transfer to which account?

[U]: To my savings account

....

[S]: The transfer has been carried out.

[S]: You can also say 'transfer some money from my current account to my savings account' to speed up the transaction.

...

The system suggests a speedier way of expressing a request when stating a similar request in the future.

Figure 7. Suggestive prompting.

When collecting user-specific interaction data it is important to evaluate the possible improvements in usability against the effort required to learn about the configuration options. Also, ‘default’ versions of the dialogue interface should be available so that users who do not want to customise their interface can still use the service.

2.5 Addressing Users

Interaction between a user and a dialogue system requires the system to address the user in various different scenarios each with a different purpose. In particular, the system needs to address the user in the following main scenarios:

- asking questions
- answering questions
- providing information

Considerations about how to address users in these distinct scenarios will be discussed in this section with a view to choosing the right wording and dialogue structure that results in a co-operative user-system dialogue.

In a final section consideration is given to dialogue interfaces for different cultures and languages.

2.5.1 Asking Questions

There are numerous different types of questions a system might need to ask a user to elicit required information. Gilbert et al. (1999) identify four major types of questions:

- navigational questions
- task selection questions
- task detail questions

- confirmation questions

Each of these types of question has particular features attached that need to be observed in order to facilitate easy information exchange between users and the dialogue system.

2.5.1.1 Issue: navigational questions and task selection questions

Task selection questions are used when information about the kind of task the user wants to accomplish is required. Such questions can be very explicit and system directed (e.g. "To obtain a balance say 'balance', to order a statement say 'statement'...") or more user-directed and not explicit at all (e.g. "What do you want to do next?"). The same distinction is often made by calling these two major types of task selection questions 'closed' and 'open'.

It is important to provide sufficient information in a task selection question so that the user can be certain about what options are available when the system asks a task selection question.

Options:

Closed questions can be used to provide all task choices that can be selected as part of the question[11].

Pros:

Sufficient information is available for the user to select a task that is supported by the dialogue system.

Cons:

Announcements that list all the possible options can become very long-winded and the cognitive load placed on the user can be too high when more than three to five options are given. This can result in the user not being sure about what they want which, in turn, can result in an unsuccessful request.

Options:

Closed questions can (and should only) be used when it can be assumed that the user knows the functionality of the system (i.e. the task options) in sufficient detail to request a task that is supported by the dialogue system. Users tend to acquire this kind of knowledge through repeated interaction with the system[12], or through using the help functionality. It should not be assumed that providing such information at the beginning of the dialogue is sufficient guidance for the user.

Pros:

Users can select a task quickly, and feel that they are in control of the dialogue. Also, since both the user and the dialogue system appear to be interacting with the same background knowledge and assumptions user satisfaction might be increased (e.g. a common understanding between the two conversational partners has been established.)

Cons:

[11] Refer to section 0 for limitations regarding this.

[12] This can be 'monitored' – refer to section 0 for a discussion.

If users are in doubt about the functionality of the system they might not know what to say. In this case they are either likely to say something that cannot be understood by the system, request the wrong task or ask for help. Provided the system can deal with such strategies intelligently, this is not a major problem. The user can get the chance to request the correct task in one of the following turns.

2.5.1.2 Issue: task detail questions

Task detail questions occur typically at a very embedded level in a dialogue. They tend to focus on very specific information such as the amount of money that should be transferred in an automated telephone banking service.

Since task detail questions are used to elicit important information they should be worded such that they prompt the user to respond in a way that is expected by the system. This is a major consideration when speech recognition technology introduces constraints on the way in which users can express themselves (e.g. rigid grammars). These constraints should be communicated to users with wording that they can easily understand so that they are in no doubt about how to respond to the question. This is the same strategy that can be adopted when the dialogue system consistently fails to collect important information from the user.

Options:

Instructions on how to respond to the question can be given as part of the question: "Please state the date when you want to leave, for example 'Thursday the 25th of February'."

Pros:

Users are aware of the system limitations and are therefore more likely to respond in a way that results in a successful next dialogue step.

Cons:

Experienced users might not need detailed instructions and might therefore not like the long system announcements.

2.5.1.3 Issue: confirmation questions

Confirmation questions are normally framed in 'normal' question form, i.e. interrogative structure, and they are asked in response to a user input. They are crucial in many services to enable 'grounding' of the interaction.

Confirmation questions are frequently used after a response to a task detail question in order to make sure that the specific piece of information was understood correctly by the system.

Options:

Confirmation questions do not need to be asked after each collected information token, but rather can be asked to confirm a complete set of information, such as money amount, source account, target account, etc. However, it is important to then provide for means of repairing the collected information by the user should it be wrong.

Pros:

The user gets a chance to find out whether the system has understood correctly what was said. It is also possible to repair the data, or even stop the system from carrying out an incorrect task.

Cons:

Excessive use of confirmation makes the user-system dialogue long and can alienate users. Confirmation should therefore only be used where it is absolutely required. For instance, when requesting the balance for one of a number of possible accounts, it is not necessary to confirm the account prior to playing out the balance since this would make the dialogue longer in both, the successful and the unsuccessful recognition. Should there be a mis-recognition, this can easily be spotted by the user when the wrong balance (and account name) is played out and the request for a balance on the required account can easily be re-submitted.

2.5.2 Answering Questions

Providing answers to users' questions can be a difficult task. Furthermore, it is crucial that not only the correct answer is provided to a user, but this needs to be done in an appropriate way so that it insures that the user can understand the answer completely and correctly. Therefore, answers should be clear and unambiguous and[13]

- use language the user is familiar with,
- provide sufficient information (as required in the current context of the interaction),
- not provide too much information,
- be correct and relevant, and
- be short.

2.5.2.1 Issue: Clear and unambiguous answers

In order to facilitate successful further interaction, it is important to provide clear and unambiguous answers. The user needs to be able to understand the answer easily so that she can prepare for the next dialogue action.

Options: use language the user is familiar with

An important way to insure that an answer is clear and easy to understand is to consider the target-user's background and expertise. This is especially important when phrasing the answer. The wording should be chosen such that it uses language the user understands (and uses in the given context).

When required, more specific phrases than the ones used by the user can be used (e.g. in date expressions). However, a clear reference should always be made to the user's request (Figure 8).

[13] These features are adapted features taken from the CODIAL support tool for co-operative dialogue design, the background of which is partly provided by the Gricean maxims (Grice, 1975). Description of the tool is provided in (Dybkjær, 1999).

…

[S]: When do you want to travel to York?

[U]: I want to travel tomorrow.

[S]: So to confirm, you want to travel from London to York, ***tomorrow***, Friday the 26th of February?

[U]: Yes, that's right

…

The system responds with a more precise specification of the date, while re-using the users 'vocabulary'.

Figure 8. Answering questions – using user's language

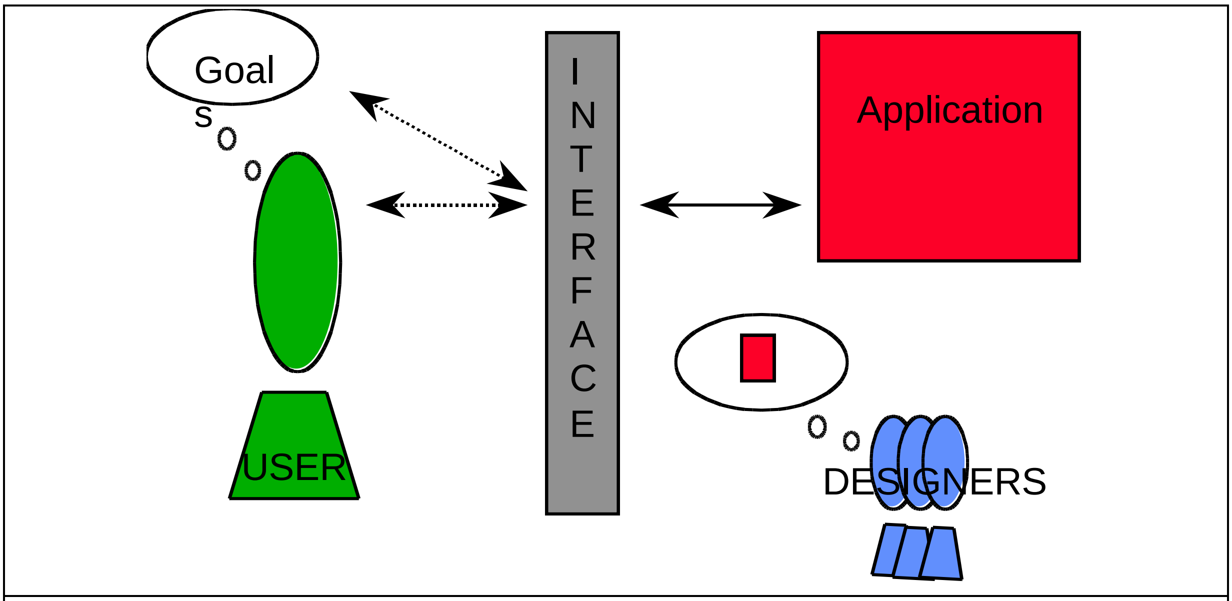
Pros:

Users do generally know the domain they are working in as well as the task they try to achieve relatively well. It cannot be assumed that they know what is behind the interface, i.e. the implementation of the dialogue system. The interface is the application for the novice user (Figure 9)!

Using users' language ensures that the information given in the answer can be understood.

Cons:

None, really – just think about it!



For the novice user the interface is the application

Figure 9. The user and the application interface

Options: Provide sufficient information

See following section (2.5.3).

Options: Do not provide too much information

See following section (2.5.3).

Options: Be correct and relevant

It is important that a user can trust a system response. Users have good reason to become irritated if the system provides false information such as incorrect departure times, prices or meeting venues. Incorrect responses can be produced because the database that holds the information has incorrect entries. However, this is not often clear to the user who can be under the impression that the incorrect information was provided because the system did not understand the user correctly. For instance, a designer told the story of an unfavourable review of an automated train timetable system by a journalist. The unfavourable review was not due to problems with interacting with the system, but rather because the system returned an incorrect time for a train connection – the underlying database was slightly out of date![14]

The lack of relevance in a systems announcement will typically lead to a need for a clarification part in the dialogue. The user will seek clarification if the relevance of a given answer is not quite clear. System irrelevance may be caused by misrecognition. In such cases it may be hard to avoid an irrelevant response (see also 2.2.3). The system's reply may be perfectly relevant given its interpretation of the user's utterance but totally irrelevant given what the user actually said.

Options: Be short

Answers provided by the system, which are too long increase the time needed to achieve a certain task. Since there is no way to speed up the information, users may become bored and inattentive. They may even try to take over the initiative if the system talks too much. System announcements that are too verbose do not affect transaction success, however the problem will be revealed through users' comments and opinions on the system (Williams and Cheepen, 1998, Bernsen et al., 1998).

2.5.3 Providing Information

One of the main functions provided by many SLDSs is to provide information such as train timetable information, information on boat traffic, etc[15]. In order to facilitate a successful and quick interaction that supports the user in achieving the task they set out to do, a number of aspects need to be considered when providing information. As already discussed above (section 2.5.2), system announcements should be short and correct. In addition to that, information should be provided in a uniform, non-ambiguous way phrased using the user’s language.

It is important to note that when crucial and/or complex information is provided the user might want to write this information down (e.g. account balances). The interface needs to make provisions for this such as allowing the user to pause the information playout, or a slower information playback that considers a concurrent task of writing the information down.

[14] Harald Aust at a presentation at the ISSD workshop in Sydney, December 1998.

[15] Another major function which is not explicitly considered here is to provide a service, e.g. Voice Activated Dialling.

2.5.3.1 Issue: Uniformity in information provision

The same type of information should always be presented in the same way using consistent vocabulary and construction (grammar) of the announcement. Users can understand provided information much more easily when it follows ‘standard’ patterns.

Options:

If complex information such as train timetable information is to be provided to the user, the various different information tokens (departure time, arrival time, etc.) should always be expressed in the same order in all system announcements.

Pros:

Using consistent ordering of information tokens in system announcements enables users to ‘skip’ information that they don’t require. Furthermore, it is possible to predict what information token is to be played out next and how much more is to come. Users are therefore able to receive and understand complex information more easily.

Cons:

A rigid structure in the information play-out might contribute to usability problems and low perceived usability by the user for a number of reasons. If the structure for information provision is not perceived as being natural and logical, a user might perceive the dialogue interface as being un-cooperative. For instance, a user of a voicemail system might prefer to learn about the sender of the voicemail and the subject before the time and date when the message was sent. However, if the information playback is structured such that information about the sender, the time and date and then the subject is announced, interaction is less usable for this user. Careful observation of the way in which users perform tasks with a system can lead to a different design that does not display this problem. Furthermore, customisation could be provided (see section 2.4.2). Also, a rigid structure for information provision might be perceived as not co-operative by the user because no obvious attempt is made to change the phrasing to suit the user’s requirements.

Options:

The vocabulary for system announcements should not change when playing out similar type of information. This, however requires that the vocabulary has been chosen correctly (see section 2.5.2).

Pros:

The user can understand quickly what the meaning of the information is without having to interpret the system announcement.

Cons:

None, unless the vocabulary has not been chosen very well.

A specific pieces of information is worth considering in more detail[16]:

[16] See also (Gilbert, et al., 1999).

2.5.3.2 Issue: Playing out number strings

Number strings are used for playing out telephone numbers, order numbers, etc. Because of their different meaning different characteristics for saying specific number strings have emerged. In order to provide users with a well-known means of referring to numerical data it is important to adopt such frequently used characteristics. However, it is important to remember that these characteristics are not easily portable to other languages and cultures (see section 2.5.4).

Options:

Gilbert, et al. (1999) point out that normal numbers in English should be played out in chunks of two (e.g. “seven six – three four – three one”). The reason for this is that this strategy follows the pattern normally used in television and radio advertisements, where the listener is not able to interrupt the information sequence to, for instance, enable writing down of the information.

However, Gilbert, et al. (1999) also point out that special characteristics (e.g. “nine nine nine”), otherwise memorable structures (e.g. “twenty twenty twenty”), or structures that have a certain visual representation (e.g. invoice numbers, part numbers, etc.) should be played out using normal conventions.

Pros:

Users are provided with potentially complex number information in a way they can deal with it easily. Memorising or writing the information down is therefore easily possible.

Cons:

Although a general rule is provided, numerous exceptions to this exist which might complicate the interface.

2.5.4 International dialogue interfaces

Spoken dialogue systems are now developed and deployed around the world, therefore it is essential that the linguistic and meta-linguistic content and logical description of tasks is matched to the target country (del Galdo and Nielsen, 1996). For instance, designers of a word-spotting based system to be deployed in the USA have to consider that Americans are known to use the phrase "no problem" to mean "yes". If a system is designed such that it only spots the words "yes" or "no" in an utterance, the system response to a "no problem" answer given by a user will be completely wrong (see Figure).

A careful analysis of the nature of the task users want to achieve by using the dialogue system has to be carried out prior to simply translating an existing application into a new language. Cultural differences might not only affect the language and phrases used for expressing commands, but also the actual task. Since there are only a very limited number of systems available that offer the same service in different languages and have been developed using the same technology[17], an in-depth study of cultural issues with respect to spoken dialogue

[17] A number of train timetable information systems have been developed in the framework of the EC-LE project ARISE. The EURESCOM P502 project MIVA (Multilingual Interactive Voice Activated telephone services) has also provided automatic multilingual telephone assistants in the telephone-service field (Leonardi, et al., 1997).

interfaces has not been carried out as part of the DISC project. Therefore, no best practice recommendations can be made.

2.5.5 Politeness

When considering the aspect of politeness in dialogue interfaces it is important to consider the context in which, and the tasks for which, the dialogue system is going to be used. The frequency of use should also be a consideration.

Options:

A dialogue system that provides a very specific service for expert users (e.g. task-driven interaction) should not use politeness tokens.

Pros:

Williams and Cheepen (1997) show that in the business-oriented application domain of telephone banking, the inclusion of linguistic tokens of friendliness and politeness tend to work against end-user satisfaction. They found that end-users in their experiment perceived dialogues which included such material as "long-winded" compared with more skeletal dialogues which do not contain such tokens.

It was also shown in separate tests that lengthy, more polite system announcements that provide extra explanation (Basson et al., 1996).

It is important to note, that the findings in these studies are likely to be specific to the cultural context and the service as well as the domain context. Only limited conclusions regarding politeness in system announcements can therefore be drawn from these studies.

Cons:

System announcements without politeness tokens may seem un-friendly to first-time users. Furthermore, the organisation offering a particular service might require a 'polite' style because the service provides a way of displaying commitment to customer service. It is therefore important to the organisation to be represented as 'friendly and caring'.

3. Introduction to Human Factors Design Life-Cycles

Thus far, this document has described individual human factors aspects of interface design for spoken language dialogue systems. However, in order to design and develop an SLDS with a user interface that displays the best practice aspects discussed, a user-centred design and implementation life-cycle has to be considered. This enables easy and quick engineering for usability of the dialogue interface.

This section provides an introduction to user-centred interactive system design by explaining the reasons for the need of a user-centred process and the ideas behind it. The following section describes such a life cycle in more detail.

3.1 Issue: Interactive systems design life cycle

The structure of the design life-cycle of spoken language dialogue systems is determined by the interactive nature of the artefacts under design. Interactivity necessitates a particular style of design for the following reasons:

- Dialogue systems are used by people and organisations who
    - have prejudices,
    - learn to interact with interfaces, and
    - display erratic behaviour.
- Dialogue systems can only be fully validated by using them.
- Dialogue systems are complex; interfaces are more complex and environments are even more complex. The designer must be aware how these different sets of factors interact.
- It is difficult to validate systems with a human component since human behaviour is highly erratic and contextualised.

Successful interactive system design focuses on representative users being included as early as possible in the design process. With this in mind, Poltrock and Grudin (1995) suggest four key aspects of interactive system design processes:

Early and continual focus on users: Designers should have direct contact with intended or actual users via interviews, observations, surveys and participatory design. The aim is to understand users' cognitive, behavioural, attitudinal and ergonomic characteristics and the characteristics of the job/task they are doing.

Early and continual user testing: The only presently feasible approach to effective user interface design is an empirical one. This involves observations and measurements of user behaviour with mock-ups or pilot systems.

Iterative design: A system must be developed in response to user tests of functions and the whole interface. Early iterations should be cheap and flexible using, e.g., text versions of voice dialogues. As the product develops, later iterations will involve users less and be on the target computing platform. The (software) tools to make interface design iteration possible must be available.

Integrated Design: All parts of system development (user interface, documentation, training) must evolve in parallel and should be under one management.

In addition to these four key aspects of the design process, continuous and comprehensive analysis and evaluation of the information gathered as well as the design process has to be performed. It is not enough to measure user characteristics etc. Care has to be taken that the right (appropriate) characteristics are measured using the correct methodology.

Case Study: The perils of Ignoring Users

A large bank decided that it would provide a speech and tone driven telephone banking service. One of the services offered was the ability to pay a bill. To do this the customer was required to nominate a bill reference number and an amount. The transaction was then confirmed and the caller was asked if they required another of the banking services on offer. This dialogue was decided without any consultation with end-users (other than the designer-employees themselves) and was implemented and deployed.

Once out in the field it appeared that the bill payment dialogue was causing the average call length to increase. The reason was not because of poor speech recognition but was related to the typical behaviours of banking callers when paying bills.

It transpired that users often wanted to pay a clutch of bills within one call. The initial dialogue design allowed this, but only after renegotiating the whole dialogue again. This proved both time consuming and frustrating for callers. In response to this user study, the dialogue was changed and users were given the option ‘Pay another Bill?’.

Case Study 4. The perils of Ignoring Users.

Currently, the majority of spoken language dialogue systems that are developed in industry do not follow the same processes as other interactive applications since they are viewed more as technical systems (technology-centred design process). There is little formal user analysis (tasks, environments) and evaluation occurs late in the product’s life. The results of the late evaluations often uncover common-sense errors which could have been corrected during the requirements capture stage of the product (See Case Study 4).

Section 4 provides a detailed description of a user-centred development and evaluation approach for SLDS design which is a re-worked version of (Bernsen et al., 1998). Gilbert et al. (1999) provide a similar approach which is focussed on the applicability and uptake of the design guidelines within an industrial context.

3.2 Issue: Design life cycle in industrial or research context

The design and development of speech systems does not occur in isolation. Rather, it is part of an organisational context which places constraints on the type of life-cycle that is possible. It is often the case that the ideal life-cycle is not adhered to due to contextual reasons. A gross distinction can be made between industrial and research environments. Important aspects, i.e. those which directly affect system design processes, of these two environments will now be discussed in detail.

The complex nature of spoken dialogue systems requires that their design and implementation impacts on a number of different skill-bases over some considerable period of time. Not least in this process is the consideration of the type of product life-cycle which is to be adopted.

An important aspect of SLDS design is that they need to be thought of also in terms of satisfying some need rather than as a piece of technology. Ideally, from a user perspective, technology has to be constrained by the interaction, and not the interaction by the available technology.

This requires easy and continued 'access' to representative users. However, this can pose a problem because developers in industry often do not wish their products to be seen by the general public in the early stages of development. This often leads to a situation where user testing is left until the beta-trial stage, when there is a much greater inertia to change. If any early testing does take place, a less representative sample is normally obtained from company employees for fear of adverse publicity. In a research environment, users will more often than not be non-representative student users.

4. Human Factors Design Life-Cycle in Detail

This chapter presents the DISC life-cycle model proposal on best practice in human factors in development and evaluation. The DISC life-cycle model views the development process as an iterative process of optimisation among a growing number of increasingly specific constraints on the artefact to be built. Human factors issues should be viewed as one set of constraints among others.

The DISC dialogue engineering life cycle model draws on a general software engineering life cycle model but is specialised to capturing the process of developing and evaluating SLDSs and components. The DISC life cycle model asks a series of fairly detailed questions most of which are related to the model in Figure 10.

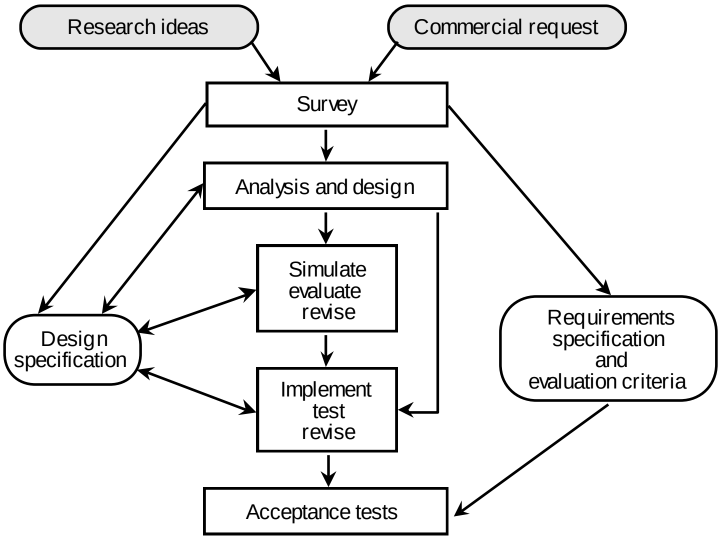


Figure 10: A software engineering life cycle model for the development and evaluation of SLDSs and components. Rectangular boxes show process phases. The development and evaluation process is iterative within each phase and across phases. Arrows linking phases indicate the overall course of the process. The requirements specifications and evaluation criteria, and the design specification (rounded white boxes) are used throughout the development process. Rounded grey boxes indicate that the system being developed may be either a research system or a commercial system. The figure is from (Bernsen et al., 1998).

Figure 0-10 shows a general software engineering life-cycle model which has been slightly specialised to the development and evaluation of SLDSs and their components. The figure shows an overall framework for the development and evaluation process through to installation at the user's site. After this point there may be maintenance of the software, the software may be ported to other platforms, or it may be adapted to new applications in which case a new life cycle starts.

The DISC life cycle model is intended to be used no matter if one describes an entire SLDS, a single component or an aspect such as human factors which cuts across systems and components. 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 precise contents of the questions to ask on each issue may, however, depend on the specific aspect. For example, the issue of evaluation is highly relevant to any aspect. However, the precise tests to carry out are very different. The human factors aspect does not relate to a particular component but is relevant as part of the development and evaluation process of any component or system and can therefore be addressed by the DISC life cycle model as well.

In the following we address the DISC life-cycle issues that are of particular relevance to human factors. The DISC life-cycle model includes a total of 33 life-cycle issues which have been roughly ordered (and numbered) according to their chronological appearance during specification, design, development and the post-development stage or according to the cluster of issues they naturally belong to. Some issues are only indirectly related to human factors concerns. Hardware constraints, for instance, normally would not appear in a list of constraints relevant to human factors despite the fact that, e.g., hardware constraints may have an effect on real-time system performance which, in its turn, may have humans factors relevance. In this sense, any life-cycle issue would appear to potentially relate to human factors concerns. However, when this is the case only accidentally, the issue is not being discussed below. But note that, by the same line of reasoning, developers should always consider whether some design decision has human factors implications.

For the sake of completeness, the (numbered) life-cycle issues which will not be discussed from the point of view of human factors below are the following, each issue being presented in terms of a label and one or more questions to the developers:

4.2 Hardware constraints: *Were there any a priori constraints on the hardware to be used in the design process?*

4.3 Software constraints: *Were there any a priori constraints on the software to be used in the design process?*

4.5 Other constraints: *Were there any other constraints on the design process?*

4.6 Design ideas: *Did the designers have any particular design ideas which they would try to realise in the design process?*

4.7 Designer preferences: *Did the designers impose any constraints on the design which were not dictated from elsewhere?*

4.8 Design process type: *What is the nature of the design process?*

4.9 Development process type: *How was the system/component developed?*

4.25 Development and evaluation process sketch: *Please summarise in a couple of pages key points of development and evaluation of the system/component. To be done by the developers.*

4.26 Component selection/design: *Describe the system/component and its origin.*

4.27 Maintenance: *How easy is the system/component to maintain, cost estimates, etc.*

4.28 Portability: *How easily can the system/component be ported?*

4.31 Property rights: *Describe the property rights situation for the system/component.*

4.32 Documentation of the design process: *Please include documents from the design process which illustrate the process and facilitate and support its understanding.*

4.33 References to additional documentation: *Please include references to any background material in which information on the human factors issues can be found.*

The DISC life-cycle model issues that were deemed of direct relevance to human factors are presented in the following.

4.1 Overall design goal(s)

What is the general purpose(s) of the design process?

Human factors issues should be included in the overall design goals for SLDS development because, at the end of the day, it is the users and their judgements that decide whether a system becomes successful or not. Overall design goal statements tend to be brief and general. However, is can make an important difference to the human-centredness, or the lack of it, of the development process whether or not human factors goals are included among the overall design goals. Including, for instance, the goal of naturalness of spoken human-system communication among the overall design goals serves as a strong reminder of maintaining constant awareness of this issue during the entire life-cycle.

4.4 Customer constraints

Which constraints does the customer (if any) impose on the system/component? Note that customer constraints may overlap with some of the other constraints. In that case, they should only be inserted once, i.e. under one type of constraint.

It is important that the customer carefully considers to add usability aspects when requirements to the system are being established, including usability performance measures and the way these should be tested at product delivery time. For research projects, there are typically no real customer. At best, the research group has contact to an organisation which potentially could be a real procurer or end-user and which is willing to provide relevant information and feedback to the project. In the case of research projects it is recommended that contact be established with potential customers to get the needed input concerning human factors.

4.10 Requirements and design specification documentation

Is one or both of these specifications documented? Describe the specifications.

Identified goals and constraints are represented in a *requirements specification* document. The purpose of this document is to eventually list all the agreed requirements, including evaluation criteria, which the envisaged system should meet. The purpose of the *design specification* is to describe and eventually make operational how to build a system which will satisfy the requirements specification and meet the evaluation criteria. The design specification, therefore, will be clearly related to, and may simply include, the requirements specification. All important parts of the requirements and design specifications should be written down.

Befitting their exploratory purpose, the framework for research systems development is often loosely defined compared to that of commercial systems. The requirements specification does not serve contractual purposes. This means that requirements can be more easily modified later in the development process because there is no procurer who has to approve of the changes made.

Human factors issues must be included among the requirements and design specifications for both industrial and research systems. This is to avoid that mistakes relating to user needs and user satisfaction are discovered only too late in the development and evaluation process.

4.11 Development process representation

Has the development process itself been explicitly represented in some way? How?

The advantages of explicit development process representations are that these can be re-used, possibly in revised form, in new projects and with new developers coming on the team, and can support re-design and maintenance. The main disadvantage is that creating explicit development process representations represents an additional cost.

A development process representation relates to a system or component and the human factors issues involved in its development. For some development processes there will be a description of initial data collection including, i.a., interviews with end-users and site visits. If Wizard of Oz experiments or later controlled tests are performed, users will sometimes be given a questionnaire and/or they will be interviewed. Data from such questionnaires or interviews will in some cases be published e.g. in scientific papers. In other cases they will be available internally in the organisation only. However, in both cases they form part of the development process representation. The reliability of data depends on how the experiments or tests were carried out and if real users were involved or if, e.g., only students were used. It is recommended to actively involve real users throughout the development process and to clearly describe in the development process representation the actions taken and the resulting information collected.

4.12 Realism criteria

Will the system/component meet real user needs, will it meet them better, in some sense to be explained (cheaper, more efficiently, faster, other), than known alternatives, is the system/component "just" meant for exploring specific possibilities (explain), other (explain)?

Realism criteria may relate to, for example, cost-saving, making the performance of a task faster and more efficient, or increased quality of service. To appropriately address real user needs, the development process has to include human factors issues such as extended end-user contact, extensive work on domain delimitation, clear adequacy evaluation criteria, and extended quantitative and qualitative evaluation throughout the development process. In research projects focus is typically on exploration. This means that realism criteria are not being considered hard constraints, that is, one may deviate from the realism criteria to the extent needed to achieve other, more important project goals. However, to develop realistic prototypes research projects need to seriously consider human factors issues.

4.13 Functionality criteria

Which functionalities should the system/component have (this entry expands the overall design goals)?

The overall function of an SLDS typically is to solve one or more tasks through a shared-goal dialogue. The functions of the components vary. For instance, the function of a speech recogniser is to recognise user input and the function of a dialogue manager usually is to control the dialogue.

Functionality criteria specify the functions to be performed by the system and its components and do not concern the usability of the ways in which those functions are performed. However, the two perspectives of functionality and usability have to be in agreement in order for the development task to be feasible. Moreover, some functionality criteria are of direct interest from a human factors perspective. These may include, e.g., that the SLDS must have a sufficient coverage of the selected task domain, be able to understand input and be able to produce reasonable answers to user input.

4.14 Usability criteria

What are the aims in terms of usability (this entry expands the overall design goals)?

Usability criteria are the central human factors issues. Since components are usually embedded in SLDSs, there are two usability aspects to consider. One relates to components for inclusion in an SLDS whilst the other relates to entire systems. The first aspect concerns the usability criteria as seen from the *system developer's* point of view. For the system developer, usability criteria typically relate to issues such as how easy it is to embed the component into the SLDS, how easy it is to maintain or modify the component, whether it will be able to run easily in different environments, and whether there is a clear and comprehensive documentation of the component.

The second aspect concerns the *end-users'* point of view of the system, which forms the focus of this report. End-user usability criteria may include, i.a., ease of use of the system, the capability of the system to perform a dialogue which is natural, flexible and robust within its domain, availability of sufficient meta-communicative facilities, sufficient task domain coverage, and awareness of the needs of different user groups (e.g. novice and expert users). All such criteria should be made operational and suitable ways of evaluating them should be specified together with target evaluation results and the points in time when evaluation will be carried out.

4.15 Organisational aspects

Will the system/component have to fit into some organisation or other, how?

Organisational aspects are relevant primarily to components developed as (commercial) stand-alone components and (commercial) SLDSs. The system or component must be useful to the organisation, fit into its environment and fit into its business programme. The purpose of a (commercial) SLDS will often be to support, partially replace, or improve a service which has traditionally been carried out by humans. Organisational aspects need thorough analysis from a human factors perspective.

4.16 Customer(s)

Who is the customer for the system/component (if any)?

For product SLDSs or components, the customer will be the organisation which buys or makes use of the system or component, typically telecoms, railway companies, banks and insurance companies, and companies that want, e.g., an automatic switchboard system. This situation is rapidly changing, however, towards a much larger variety of customers who, for instance, may want SLDSs for their Internet pages. Research system developers should link up with potentially interested customers.

From a human factors perspective having a customer, even only a potentially interested one, means that there is an opportunity to have access to end-users and knowledge about the organisation into which the SLDS or component has to fit.

4.17 Users

Who are the intended users of the system/component?

The users of SLDS components are mainly system developers using the components for SLDS applications. The users of entire SLDSs are often the broad public. For example, the person calling a switchboard system, a telephone service or a train timetable information system may be anybody. Thus, many SLDSs are walk-up-and-use systems requiring no particular skills in advance. The main restriction often is the particular language used as SLDSs are not language independent.

From a human factors perspective it is essential to gather knowledge about the intended users.

4.18 Developers

How many people took significant part in the development? What were their backgrounds?

Often, several developers are involved with human factors issues simply because human factors cut across all system or component development and evaluation. The backgrounds of the developers may vary considerably, including engineering, computer science, linguistics and psychology. The important point is to have someone on the development team whose job it is to make sure that human factors issues are properly taken care of from the outset and throughout the development process. It will usually be preferable that people involved in human factors issues not only have experience in usability but also have the relevant technical knowledge that can ensure that what the development team undertakes to do is realistic.

4.19 Development time

When was the system/component developed? What was the actual development time for the system/component (estimated in person/months)? Was that more or less than planned? Why?

Development time is system or component related. Human factors issues are only part of what it takes to build an SLDS or component. It must be kept in mind, however, that the taking into account of human factors issues may include time-consuming trial-and-error experimentation and comparatively little routine-based work.

4.20 Requirements and design specification evaluation

Were the requirements and/or design specifications themselves subjected to evaluation in some way, prior to system/component implementation? If so, how?

Human factors issues should be considered both in the requirements and in the design specification and should be evaluated along with all other parts of the two specifications. However difficult this may be to do in any formal way, it is essential to good development practice to carry out a systematic and explicit evaluation of whether the goals and constraints are reasonable, covering, feasible and non-contradictory.

4.21 Evaluation criteria

Which quantitative and qualitative performance measures should the system/component satisfy?

All evaluation criteria that will be used in evaluating the final SLDS or component, including those relating to human factors issues, should in principle be specified along with the requirements specification. The evaluation criteria state the parameters that should be measured or otherwise evaluated and the results that should be achieved for the final system or component to be acceptable.

For industrial projects, the main evaluation criteria may be related primarily to cost. The really interesting figures are, e.g., the cost-per-ticket-sold, the cost-per-contract-signed, or the cost-per-customer-informed. These quantities can be measured for both human-human and human-machine versions of the interaction. They capture all aspects of the interaction, including human factors, because a less-than-fully-usable system will sell less tickets, make fewer contracts signed or keep fewer customers informed. However, these criteria are difficult or impossible to reliably apply before the system has been deployed in the field.

There is no agreed set of human factors evaluation criteria. In many projects, human factors evaluation criteria are invented as the project proceeds and no decisions are made up front concerning which human factors criteria the final system or component has to satisfy. This approach cannot be recommended because it means that developers have little support during development in terms of criteria that the system or component should fulfil in order to be considered satisfactory and acceptable by users. The definition, from early on in the development process, 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 the development and evaluation process.

Section 5 presents a set of human factors evaluation criteria which in no way is claimed to be exhaustive. While not all of the criteria are applicable to all systems or components, each criterion partially addresses the adequacy of some systems or components. They need not all be included among the evaluation criteria stated in the requirements specification. Even if not included, they may still be useful for evaluating how good or bad the system or component is and what progress is being made during its development.

The evaluation criteria in Section 5 provide a good illustration of the complexity of human factors evaluation. Few of those evaluation criteria are straightforwardly quantitative, at least for the time being. Very many of the criteria are qualitative. Several of these, and not the least important ones, are subjective and must, at least partly, be measured through interviews, questionnaires and other contacts with the users to elicit their subjective impressions from interacting with the system. Human factors evaluation criteria are often difficult to interpret because of their qualitative nature and because they are often related to users' subjective opinions. Still, qualitative and subjective evaluations are strictly needed with respect to human factors as no known quantitative measures can capture the way the system is perceived by its users.

Human factors evaluation criteria relate to the system as a whole. Thus, if a particular part of the system does not perform too well there may be other system parts which make up for the bad performance so that the overall evaluation of the system will be fairly positive. A good meta-communication strategy, for instance, may solve problems arising from the insufficiency of the speech or language processing done by the system.

4.22 Evaluation

At which stages during design and development was the system/component subjected to testing/evaluation? How (type of test, number of users, etc.)? Describe the results. How was data collection? Was data annotation done? How? Which information has been extracted from the data? What were the results? Is test material/data/test suites (and/or a description of the test conditions) available? Can the test be replicated? Can anybody perform the tests? Is anything stated about comparability of the test(result)s with those of other systems/components of similar scope?

Evaluation is, or should be, tightly interwoven with systems development. Evaluation is constantly needed throughout development to measure progress towards the goals which the system or component has to meet. However, evaluation of human factors issues in SLDSs and components is not yet fully developed in terms of established standards and procedures for best practice. There is no consensus on terminology in the field of evaluation in general. In the following we briefly present some useful distinctions.

(Bernsen et al., 1998), (Hirschmann and Thompson, 1996) and (Gibbon et al., 1997), among others, distinguish among three types of evaluation. Although clearly not orthogonal, these three types cover the relevant aspects of evaluation. Each type may be used at any stage during SLDS or component development:

- *performance evaluation*, i.e. measurements of the performance of the SLDS 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 and are not central to human factors evaluation. Progress evaluation is used to compare two iterations of the same system or component during development, such as two Wizard of Oz iterations of a simulated SLDS.

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.

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, percentage 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 refer to aspects of the context of the evaluation, such as the users involved, whether or not scenarios are being used, or whether the system being tested is an implemented one or a simulated one. 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.

Viewed from a human factors perspective, adequacy evaluation is the central type of evaluation. However, both performance evaluation and diagnostic evaluation typically provide results which are informative as regards the usability of the system or component.

Central issues to human factors evaluation include, i.a., the type of test, at which time in the process it should be performed, what type of system the test is based on (e.g. mock-up, simulation, implemented system), how many and what kind of users should be involved, under which conditions data should be collected, how the data should be analysed, and how results may affect the development process.

4.23 Mastery of the development and evaluation process

Of which parts of the process did the team have sufficient mastery in advance? Of which parts didn't it have such mastery?

In many projects, developers of SLDSs and components have little mastery of human factors issues. There are, of course, teams with a good deal of experience in the field. It is strongly recommended to include skills in human factors issues from early on in the development and evaluation process in order to avoid a high risk of later redesign because human factors issues were left largely unattended or treated wrongly.

4.24 Problems during development and evaluation

Were there any major problems during development and evaluation? Describe these.

Human factors-related problems during development and evaluation include: the developers have little experience in human factors issues; users were not sufficiently involved in the early development phases leading to substantial redesign at a late stage to make the system or component fit user needs or demands; the requirements specification did not include usability constraints, leaving the developers in the dark with respect to when to consider usability issues and which issues to focus on; the requirements specification was not properly evaluated to begin with and it turns out that there is discrepancy between technical feasibility and usability requirements, resulting in re-specification and redesign.

4.29 Modifications

What is required if the system/component is to be modified?

It is important to note that some modifications of systems or components can have significant implications for the user's interaction with the system and the tasks s/he can solve. Human factors issues should be taken into account when such modifications are made. When it comes to usability, it is often dangerous to have an identified usability problem with the implemented system and then to adopt the first idea which comes to mind for its repair. The idea may easily make matters worse.

4.30 Additions, customisation

Has customisation of the system/component been attempted or carried out? Is there a strategy for resource updates? Is there a tool to enforce that the optimal sequence of update steps is followed?

Today, many SLDSs and components are built as one-of-a-kind and are not meant for later additions or customisation. However, more generic systems and components are spreading in the increasing market. This makes it important to know which issues, including human factors issues, may arise from making additions or customisations. If, e.g., a new task is being added to an existing system, and even if the user group is still the same as for the original system, the users should be involved again. It is important to know their preferences and habits as regards the new task and how the extension will fit into their environment and their interaction with the existing system. The same applies when the system is being customised for a different language. Even if the user group remains the same in principle, the new language and its underlying culture may make a difference with respect to, e.g., how the system should express itself.

5. Evaluation Criteria

This section expands Point 4.21 from Section 4 above by presenting a set of best practice human factors evaluation criteria. Each criterion is described by reference to the draft DISC evaluation template presented in Appendix 2. The evaluation criteria have been generated on the basis of the human factors grid analysis in Section 2. The order of presentation of the criteria roughly corresponds to that of the analysis in Section 2.

5.1 System communication about the communication (meta-communication)

A. What is being evaluated (property): Which are the system's meta-communication facilities? Are these sufficient to the task(s) addressed? Is the way the system communicates with the user about the communication itself appropriate? Is the communication clear, adequate, sufficient, comprehensible, relevant and otherwise acceptable to the intended users? The communication may include help facilities (context-dependent or not), repair facilities in case of misrecognition and failure to understand the user, and clarification facilities to handle problems in understanding the user.

B. System part evaluated: Dialogue management and system output (in particular with respect to help).

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Interaction model walkthroughs and mock-up interaction analysis possibly using scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Scenarios can be specifically designed to provoke meta-communication. Analysis of questionnaires and interviews.

E. Symptoms to look for: Interaction problems and how they are handled by the system. Users asking clarification questions. Users complaining that they did not get the help or support they needed from the system or that correction of system misunderstandings was difficult.

F. Life-cycle phase(s): Appropriateness of meta-communication facilities should be tested from early interaction design and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. The main phases are simulation and evaluation of implemented version.

G. Importance of evaluation: Important. Inappropriate system meta-communication may seriously damage transaction success and user satisfaction. Much can be done to make the system's communication about the task as cooperative as possible but the need for meta-communication cannot be totally avoided. It is important to evaluate how serious and damaging a given problem is and trade it off against its frequency of occurrence.

H. Difficulty of evaluation: Difficult for non-experts in usability and without deep knowledge of the task except when a large data set gathered from interaction with real users is available. Note that meta-communication evaluation is difficult to do in Wizard of Oz simulations because the wizard is liable to understand the users too well. The DISC cooperativity tool (http://www.nis.sdu.dk/~laila/dialogue) can help non-experts in usability to detect meta-communication design errors before these are detected in actual deployment. Meta-communication is not a magical facility which can solve all problems created by an otherwise mediocre system design. In many cases, difficulties with the system's meta-communication abilities should be solved at the task communication level, i.e. by improving the way it communicates about the task.

I. Cost of evaluation: Depends on the complexity of the meta-communication facilities offered. If, e.g., context-dependent help is offered, analysis of more dialogue situations may be needed than if the same help message is provided in all situations, and data analysis is costly. The DISC cooperativity tool helps reduce the cost by supporting early interaction design and evaluation of meta-communication.

5.2 Dialogue initiative

A. What is being evaluated (property): Is the distribution of initiative between user and system appropriate to the interactive user-system task? Initiative may be with the system, with the user or a mixture of both.

B. System part evaluated: Dialogue management.

C. Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Interaction model walkthroughs and mock-up interaction analysis possibly using scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Analysis of questionnaires and interviews.

E. Symptoms to look for: Users taking the initiative when they are not expected to do so. Users apparently not knowing what to do when the system takes no initiative. The evaluator should keep in mind that mixed initiative dialogue is not always "the best" solution. Depending on the task, user-driven or system-driven dialogue may be perfectly satisfactory.

F. Life-cycle phase(s): Adequacy of dialogue initiative should be tested in early design, simulation, evaluation of implemented version, field evaluation, final evaluation. The main phases should be early design and simulation/evaluation of implemented version.

G. Importance of evaluation: Important. Inadequate dialogue initiative distribution is likely to lead to problems in communication as well as to dissatisfied users.

H. Difficulty of evaluation: Many tasks of low complexity are fairly easy to evaluate. The difficult cases are those in which one form of initiative does quite well except at certain crucial points. An example is system directed dialogue in which users are sometimes liable to take the initiative and ask questions of the system in order to be able to answer the system's questions.

I. Cost of evaluation: Medium expensive. Usability evaluation based on analysis of a relatively small set of recorded dialogues together with interviews and/or questionnaires is required.

5.3 Dialogue structure

A. What is being evaluated (property): The adequacy of the interaction structure offered by the system. This includes evaluation of issues such as whether users get access to the right functionalities at the right time, whether the access is sufficiently direct, whether there are short-cut possibilities for more experienced users and whether the interaction design supports an appropriate interaction speed.

B. System part evaluated: Dialogue management.

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Interaction model walkthroughs and mock-up interaction analysis possibly using scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Analysis of questionnaires and interviews.

E. Symptoms to look for: Many turns needed to arrive at a goal which is deep down in a hierarchy. Users having problems finding the most direct way to carry out a task. Users complaining that it takes too many steps to do a task or that it is difficult to find out how to get a certain task done. A sequence of questions to users which is illogical or counter-intuitive from the users' point of view.

F. Life-cycle phase(s): Dialogue structure design should be tested from early design and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. The main phases should be early design and simulation/evaluation of implemented version.

G. Importance of evaluation: Important. Inadequate dialogue structure design is likely to lead to unnecessarily lengthy and error-prone communication as well as to dissatisfied users.

H. Difficulty of evaluation: Not too difficult to do for someone who (a) knows the domain and the task very well, and (b) has access to a sample of end-user dialogues with the system. Both (a) and (b) are mandatory for proper evaluation. However, it is also necessary to ask users their opinion in order to be able to judge properly whether the dialogue structure design is adequate.

I. Cost of evaluation: Standard for usability evaluation based on a relatively small set of recorded dialogues together with interviews and/or questionnaires. The more complex the dialogue structure, the higher the cost of making sure that it is the right one.

5.4 Modality appropriateness

A. What is being evaluated (property): A modality is a way of exchanging information between users and computer systems, such as input speech or output graphics text or images. Is speech input and/or output an appropriate modality for the application? Is the combination of speech with other modalities appropriate? Are there better solutions than the input/output modality combination which was chosen for the application? Is the use of speech, possibly combined with other input/output modalities, appropriate and supports natural interaction with the system.

B. System part evaluated: System input and output.

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Use of the DISC speech functionality tool (Bernsen and Luz, 1999; http://disc.nis.sdu.dk/smalto). Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Analysis of questionnaires and interviews.

E. Symptoms to look for: Interaction problems caused by the use of inappropriate modalities. Modalities which are available but are not being used.

F. Life-cycle phase(s): General modality appropriateness should be thoroughly evaluated in early design. The DISC speech functionality tool can be used when the system specification is available. Evaluation of the more detailed way in which the chosen modalities (or modality) are being used for the exchange of information between user and system, can be tested with users when a first simulated or implemented version of the system is available and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. Main phases are early design, simulation and evaluation of implemented version.

G. Importance of evaluation: Important for innovative applications. Inappropriate modality choice can make the application useless, awkward or unnecessarily annoying depending on the severity of the mistakes made in choosing the modalities. Only if the system to be built is very close in all respects to existing successful applications, can it be assumed that the modality choice is a correct one. In all other cases, the modality choice should be evaluated.

H. Difficulty of evaluation: The question of what speech is or is not good for, and with which other input/output modalities speech can profitably be combined, is a complex one. Even if the system only uses speech it should be evaluated whether speech is appropriate for solving the users' tasks. For instance, speech is not well-suited for giving the user a quick overview of a long list of boat departures. Graphics is much better for this purpose. The difficulty of evaluation lies in becoming aware of other than the most obvious cases of modality inappropriateness. The DISC speech functionality tool is the only available early design decision support tool. Experts in the field are hard to find. Even scientific empirical experimentation may in some cases be necessary before launching the development project.

I. Cost of evaluation: The DISC speech functionality tool is a low-cost means of understanding the issues to be faced when applying speech-only or speech together with other modalities to novel applications. It is recommended that the more detailed evaluation be based on analysis of a limited set of recorded dialogues/interaction sessions together with interviews and/or questionnaires. The cost of this is likely to be modest-to-medium depending on the complexity of the system and the modalities involved.

5.5 System communication about the task

A. What is being evaluated (property): The appropriateness of the way the system communicates with the user about the common task. Is the information provided clear, adequate, sufficient, comprehensible, relevant, well-ordered and otherwise acceptable to the intended users?

B. System part evaluated: Inappropriate system output may be caused by several SLDS modules. For instance, the recogniser or the language understanding module may have misrecognised or misunderstood the user's input leading eventually to irrelevant system output. But even when these modules work properly, the output design itself and the speech generation play a crucial role in successful system task communication. Ambiguous system information, for instance, or inadequate feedback or low quality synthesis, can easily corrupt the interaction.

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Interaction model walkthroughs and mock-up interaction analysis possibly using scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Analysis of questionnaires and interviews.

E. Symptoms to look for: There are very many things to look for, including the following. Mismatches between user input and system output. Lack of system understanding. Missing feedback. Too little informative system output. Ungrammatical, ambiguous or obscure system output. Lengthy system output. Users complaining about the system speaking too much or being somewhat rude (impolite). Disturbing pronunciations of system output.

F. Life-cycle phase(s): Should be done from early interaction design and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. The main phase is early design and to some extent simulation/evaluation of implemented version.

G. Importance of evaluation: Important. Inappropriate output to the user may seriously damage transaction success and user satisfaction. However, it should be noted that too human-like speech output may cause users to interact with the system in a way which is closer to human-human interaction than what the system can actually handle.

H. Difficulty of evaluation: Difficult for non-experts in usability and without deep knowledge of the task except when a large data set gathered from interaction with real users is properly analysed. Proper evaluation normally requires someone who (a) knows the domain and the task very well, and (b) has access to a sample of end-user dialogues with the system. The DISC cooperativity tool can help non-experts in usability to detect dialogue design errors before these are detected in actual deployment.

I. Cost of evaluation: If used correctly the DISC cooperativity tool can do most of the work and thus helps reduce the cost considerably. Medium expensive if also usability evaluation based on analysis of a relatively small set of recorded dialogues together with interviews and/or questionnaires is involved.

5.6 Adaptivity to user differences

A. What is being evaluated (property): Does the system need to respond in a flexible manner to different groups of users, such as novice and expert users? Does it adequately do so?

B. System part evaluated: Dialogue management.

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Analysis of interaction model walkthroughs or mock-up interactions which were possibly done on the basis of scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Analysis of questionnaires and interviews. Analysis of realistic data on user task behaviour is mandatory.

E. Symptoms to look for: Interaction problems, users complaining that task completion takes too much time, involves unnecessarily many steps or that the system is confusing to interact with because it is not clear what to do.

F. Life-cycle phase(s): Should be tested as soon as a first interaction model is in place and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. Main phases are simulation and evaluation of implemented version.

G. Importance of evaluation: Important. Most systems need to be transparent to first-time users. Many of these systems can benefit from offering simple shortcuts for experienced users. For instance, novice users will typically need an explanation on how to use the system whereas expert users would like to skip that because they already know. Some systems need more, such as adaptation to individual users. Lack of attention to different user groups may easily lead to interaction problems and frustrated users.

H. Difficulty of evaluation: Can be difficult depending on the task and the intended user groups. In general, it is always difficult to judge whether the system is sufficiently flexible.

I. Cost of evaluation: Medium cost should be expected because of the need for analysing data from user-system interactions as well as from questionnaires and interviews.

5.7 Task and domain coverage

A. What is being evaluated (property): The system's coverage of the task and the domain it aims to handle. Complete task coverage means that whatever the user might meaningfully want to do to solve the task, has been taken into account by the developers. If something the user might meaningfully want to do to solve the task has been deliberately excluded, the user should have been given full, comprehensible and practiceable (memorable) information about it. Complete domain coverage means that any domain fact which the user may meaningfully want to access, is actually accessible to the user. If domain facts which the user might meaningfully want to access to solve the task have been deliberately excluded, the user should have been given full, comprehensible and practiceable (memorable) information about them. In other words, task and domain coverage evaluation must take into account both the nature of the task in itself, and the task and domain coverage as announced to users by the system. Even then, the developers may have made erroneous decisions about which task and domain parts to exclude. This is the case when standard user expectations are that these items are actually included. In such cases, users may judge the system negatively whatever its announcements.

B. System part evaluated: Lack of task and domain coverage may relate to any part of an SLDS.

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Analysis of interaction model walkthroughs or mock-up interactions which were possibly done on the basis of scenarios. WOZ data analysis. Analysis of data from user interaction with the implemented system. It is a good idea to construct scenarios because this helps think about details of the task and the domain. Analysis of questionnaires and interviews. Analysis of realistic data on user task behaviour is mandatory.

E. Symptoms to look for: User requests which are clearly within the task domain but to which the system does not produce a reasonable answer. There may be other reasons for inadequate system output than lack of task and domain coverage, e.g. recognition errors. However, if such communication problems occur it should be checked if the system actually covers the task domain sufficiently. Missing vocabulary and grammar coverage may also be a symptom of inadequate task and domain coverage.

F. Life-cycle phase(s): Should be done from early interaction design and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation, maintenance. Main phases are early design and simulation/evaluation of implemented version.

G. Importance of evaluation: Important. It is important that the system adequately covers the task(s) and the domain it claims to handle. Lack of coverage may easily lead to serious break-downs in user-system interaction.

H. Difficulty of evaluation: Can be difficult, especially for task coverage in all but the simplest systems. Users often have their own models of the task. If the system's task model is restricted compared to the users' model(s), strong system explanations of those differences are needed, and even these may not always be enough. Moreover, even task experts can be taken by surprise by finding that users want to do perfectly relevant sub-tasks which they (the experts) did not anticipate. Deep task knowledge is needed for the evaluation.

I. Cost of evaluation: Depends very much on the task and, to a lesser extent, on the domain. Some cost is unavoidable because of the need for data gathering and analysis.

5.8 Coverage of user vocabulary and grammar

A. What is being evaluated (property): How well does the system's vocabulary and grammar cover user input? Does the system cover its own output vocabulary and grammar? Ideally, the system's vocabulary and grammar are adequate for processing any task-relevant user input. In practice, however, and except for very simple systems, there will be gaps to be discovered during deployment of the system.

B. System part evaluated: System vocabulary and grammar typically form part of the recognition module and/or of the language understanding module. Some systems have a vocabulary and a grammar for each of these two modules.

C: Type of evaluation: Quantitative, qualitative. Qualitative evaluation is needed because it is hard to count grammatical structures or rules. It is also hard to judge their relative significance. And for vocabulary, the question whether, e.g., "lillejuleaften" should be in the vocabulary or not, is a qualitative question. Even if one user did use it, it might be excluded.

D. Method(s) of evaluation: Design analysis. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Logfile inspection. Output from a program which will compare all words in the user's utterances to those included in the systems vocabulary and list those which were not found.

E. Symptoms to look for: Mismatches between user input and system output, and lack of understanding by the system might be symptoms of missing vocabulary or grammar but might as well indicate other problems. Logfiles showing recognition and/or parser errors.

F. Life-cycle phase(s): Should be done as soon as it is possible to have users interact with the system and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation, maintenance. Main phases are early design and simulation/evaluation of implemented version. More reliable data will be obtained with an implemented than with a simulated version of the system because the uncertainties which may be caused by the wizard's performance will be avoided.

G. Importance of evaluation: Important. It is important to achieve a close-to complete vocabulary and grammar coverage. The lower the coverage the more interaction problems are likely to occur.

H. Difficulty of evaluation: It is straightforward to find words and constructs out of system vocabulary or grammar in a transcribed corpus.

I. Cost of evaluation: The cost of finding words and constructs out of system vocabulary or grammar in a corpus is relatively low because part of the process can be automated, cf. point D.

5.9 Information about system capabilities, limitations and operations

A. What is being evaluated (property): Are the users being told what the system can and cannot do for them and how they should operate the system? How (on-line, off-line, both, only at the start of the interaction or also at certain points during the dialogue)? Is the information provided adequate, sufficient, comprehensible, timely and otherwise acceptable to the intended users? The information may include, i.a., which task(s) the system can solve and whether it has any peculiar characteristics, such as not listening while it talks or only accepting single words as input. Note that it is not sufficient that the system comes up with a list of issues which the users must take into account. It must also be possible for the users to do in practice what they are told.

B. System part evaluated: It is part of output generation to phrase what the system can and cannot do and how one should interact with it, and it is part of dialogue management to decide when to address the users.

C: Type of evaluation: Qualitative, subjective.

D. Method(s) of evaluation: Interaction model walkthroughs and mock-up interaction analysis possibly using scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Analysis of questionnaires and interviews.

E. Symptoms to look for: Interaction problems, users complaining about unnatural interaction, lack of clarity about navigation or lack of clarity of what the system actually can do for them.

F. Life-cycle phase(s): This criterion should be applied from early interaction design and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. Main phases are early design and simulation/evaluation of implemented version. It is very important to confront the intended user population(s) very early on.

G. Importance of evaluation: Important. Unclear or missing communication of what the system can and cannot do and how the user should interact with it is likely to lead to communication problems, failed dialogues and dissatisfied users.

H. Difficulty of evaluation: The evaluation task is difficult but not enormous. With real users involved during early design, and careful study of the data gathered from their interactions with the designed system, it is possible for a usability expert to prevent most of the problems. The DISC cooperativity tool can help non-experts in usability. A difficult case is problems which may occur frequently without serious consequence but which sometimes do cause a serious problem. For instance, several systems do not listen while they talk. In many cases this does not matter even if users talk while the system speaks because users mainly just say ‘yes’ or ‘no’ to show that they follow the conversation. However, if a user tries to make, e.g., a correction while the system talks this will not be captured by the system and a communication problem is likely to occur.

I. Cost of evaluation: Low-cost evaluation may be done by analysing, e.g., 15 dialogues. The more dialogues (and questionnaires/interviews) analysed the more expensive the evaluation will be. However, more dialogues, up to a certain point, also means increased reliability of having found most or all problems. Dialogues in which interaction problems were experienced are good candidates for evaluation.

5.10 Number of interaction problems

A. What is being evaluated (property): The number of interaction problems occurring in dialogues with the system. An interaction problem is operationally defined as any potential or actual violation of the guidelines for cooperative system behaviour (Bernsen et al., 1998).

B. System part evaluated: Interaction problems may be caused by any SLDS component.

C: Type of evaluation: Quantitative, qualitative, subjective.

D. Method(s) of evaluation: Interaction model walkthroughs and mock-up interaction analysis possibly using scenarios. Wizard of Oz data analysis. Analysis of data from user interaction with the implemented system. Logfile inspection for diagnostic analysis. Analysis of questionnaires and interviews.

E. Symptoms to look for: Actual and potential mismatches between user input and system output and vice versa, lack of system understanding. Potential mismatches are such which can be detected directly from the way the system expresses itself. If the mismatch is caused by the system, it is an interaction problem to be addressed by the developers. Complaints in questionnaires and interviews.

F. Life-cycle phase(s): Should be done from early interaction design and onwards. Early design, simulation, evaluation of implemented version, field evaluation, final evaluation. Main phases are early design, simulation and evaluation of implemented version. It is very important to get rid of as many interaction problems as possible very early in the process, cf. point I.

G. Importance of evaluation: Important. A system which causes many different, or serious, interaction problems is not user-friendly and may be impossible to use in practice.

H. Difficulty of evaluation: Difficult even for modest-complexity dialogues except when the evaluator has access to a very large and representative corpus of recorded user-system interactions. Without such data, it is often difficult to detect potential problems which are likely to lead to interaction problems under slightly different conditions. The DISC cooperativity tool can help non-experts in usability to detect interaction problems before these are detected in actual deployment.

I. Cost of evaluation: It is costly to do an extensive and systematic analysis of corpus data from users' interaction with the implemented system, and it can be costly to revise the system based on the analysis. The economic approach is to try to get rid of as many interaction problems as possible before implementation, e.g. by applying the DISC cooperativity tool.

5.11 User satisfaction

A. What is being evaluated (property): How satisfactory does the user find the system. User satisfaction is a complex intuitive product of very many perceived properties of the system, such as: how easy is it to solve a task? Does the system make many or few errors? How easy is it to correct misunderstandings? How easy is the system's output to understand? Can the user express himself naturally and still be understood by the system? How well prepared was the user to use the system? Would the user prefer the system or a human interlocutor if he had the choice? How does the user like the system? Is the system useful as it is now? If not, would it be useful with some modifications? Which changes would the user like to propose? Is the system rigid, flexible, stimulating, boring, (in-)efficient, (un-)desirable, (un-)reliable, complicate, simple, (im-)polite, (un-)predictable, fast, slow, (un-)acceptable.

B. System part evaluated: User satisfaction may relate to any part of the system and its context of use.

C: Type of evaluation: Subjective.

D. Method(s) of evaluation: Analysis of questionnaires and interviews. Despite the lack of standards, these methods can give crucial insights into what should be revised in order to improve user satisfaction.

E. Symptoms to look for: User complaints.

F. Life-cycle phase(s): . User satisfaction can be tested once a (simulated) system is available and onwards. Simulation, evaluation of implemented version, field evaluation, final evaluation. Main phases are evaluation of implemented version, field evaluation, and final evaluation. The most reliable results will be obtained with the fully implemented system.

G. Importance of evaluation: Crucial. If users are dissatisfied with the system they may not use it. It they have to use it, it may slow down their work. Even if we cannot make any safe predictions on how users will react to and evaluate the final system once it is being deployed, it is still very important to get empirical data and evaluate user satisfaction during development.

H. Difficulty of evaluation: The difficulty is to find out if users are sufficiently satisfied before deploying the system. In-house evaluation with a small number of non-representative users is insufficient. Evaluation should always involve representative users from the target user group(s). It is moreover recommended to deploy the system in the field during an experimental phase in which users will be told that the system is preliminary and that they can help improving it. In return for their help, users could be offered free or reduced-cost service. Analysis of the data is hampered by the lack of standards noted above as well as by the lack of knowledge about the multitude of factors influencing users' opinions.

I. Cost of evaluation: Costly because of the amount of data to be gathered and analysed.

6. Conclusion

This report has described a draft proposal on best practice methods and procedures with respect to human factors-related aspects of commercial and research spoken language dialogue systems. Section 2 discussed human factors grid issues, best practice options and their pros and cons. Sections 3 through 5 discussed life-cycle issues, presenting proposals on how to incorporate human factors in the development and evaluation process of SLDSs and components.

The next steps are (a) to thoroughly test the draft best practice proposal presented in this report and (b) package the resulting version for ease of access to the target user groups. These are major goals of DISC-2.

Acknowledgements

Many thanks go to the DISC partners who have demonstrated their systems in order to provide information for the exemplar analysis grids and life-cycles.

References

Basson Sara, Stephen Springer, Cynthia Fong, Hong Leung, Ed Man, Michele Olson, John Pitrelli, Ranvir Singh, Suk Wong (1996), 'User participation and compliance in speech automated telecommunications applications', In *Proceedings of ICSLP 96*, Philadelphia, pp. 1680-1683.

Bertenstam, M. (1995). 'The Waxholm system - a progress report.' In *Proceedings of the ESCA Tutorial and Research Workshop on Spoken Dialogue Systems*. Vigsø,

Bernsen, N.O.: Towards a tool for predicting speech functionality. *Speech Communication* 23, 1997, 181-210.

Bernsen, N.O.: Working Paper on Dialogue Management Evaluation. DISC deliverable D3.10, April 1999.

Bernsen, N. O., L. Dybkjær and H. Dybkjær (1998) 'Designing Interactive Speech Systems - From First Ideas to User Testing'. Springer Verlag 1998.

Bernsen, N.O. and Luz, S.: SMALTO: Speech Functionality Advisory Tool. DISC deliverable D3.9, April 1999.

Brewster, S. A., P. C. Wright, A. D. N. Edwards (1994) 'The Design and Evaluation of an Auditory Enhanced Scrollbar'. *In Proc. of SIGCHI'94. ACM Press. pp 173-179.*

Cheepen, C. (1996) 'Designing advanced voice dialogues - what do designers do and what does this mean for the future?', *http://www.soc.surrey.ac.uk/research/reports*

Cheepen, C. and Monaghan, J. (1997) 'Dialogue Design: Confirmation in Transactional Telephone Dialogues and the Problem of Yes and No'. *In Proc. of 1st International Workshop on Human Communication, Bellagio, Italy*. July.

del Galdo, E. M.; J, Nielsen, (1996). 'International User Interfaces', (New York: Wiley).

Dutton, D, C. Kamm, S. Boyce (1997) 'Recall Memory for Earcons' *Proc. of EuroSpeech'97.*

Dybkjær, Laila (1999) 'Specification of CODIAL, a tool in support of co-operative dialogue design'. DISC deliverable D2.8, February 1999.

Failenschmid, Klaus and Lin Chase (1999) 'Draft Proposal on Best Practice Methods and Procedures in Systems Integration', Disc deliverable D3.7, February 1999.

Falzon, P. (1990) 'Human-computer interaction: Lessons from human-human communication' In *Cognitive Ergonomics, Understanding, Learning and Designing: Human Computer Interaction*, Academic Press Ltd.

Falzon, P. (1985) 'The Analysis and Understanding of an Operative Language', in B. Schackel (ed), *Proc. of InterACT'85,* pp 437-441.

Franzke, M. (1997) 'Is Speech Recognition Usable? An Exploration of a Speech based Voice Mail System' In SIGCHI Bulletin, Vol. 25(3), ACM Press, pp 49-51.

Gibbon Dafydd, Roger Moore and Richard Winski (1997). '*Handbook of standards and resources for spoken language systems*.' Mouton de Gruyter. Berlin, New York.

Gilbert, Nigel, Christine Cheepen, Klaus Failenschmid and David Williams (1999) 'Guidelines for Advanced Spoken Dialogue Design', *http://www.soc.surrey.ac.uk/research/guidelines.*

Grice, P. (1975) 'Logic and conversation'. In P. Cole and J.L.Morgan, (eds.*) 'Syntax and Semantics*', Vol. 3, Speech Acts, New York, Academic Press, 1975, 41-58. Reprinted in P. Grice (1989). '*Studies in the Way of Words*'. Harvard University Press, Cambridge MA, 1989.

Gustafson, J., A. Larsson, R. Carlson and K. Hellman (1997) 'How do System Questions Influence Lexical Choices in User Answers' In *Proceedings of EuroSpeech'97*, Rhodes, 1997, pp 2275-2278.

Hone, K., C. Baber (1995) 'Using Simulation to Predict the Transaction Time Effects of Applying Alternative Levels of Constraint to User Utterances with Speech Interactive Dialogues' *In Proc. of the ESCA Workshop on Dialogue Systems, Vigo, .pp 209-212.*

Jack, M (1996) 'An Investigation of Menu Strategies' Report of Dialogues 2000 Projects, CCIR, Edinburgh University.

James, F. (1997)'AHA: Audio HTML Access' In Proc. of 6th WWW Conference, http://www6.nttlabs.com/HyperNews/get/PAPER296.html

Leonardi Fulvio, Giorgio Micca, Sheyla Militello, Mario Nigra (1997), 'Preliminary results of a multilingual interactive voice activated telephone service for people-on-the-move' In *Proceedings of Eurospeech '97*, Rhodes, 1997, pp. 1771-1774.

Nielsen, Jakob (1993), *Usability Engineering*, Boston: Academic Press Professional

Norman, Donald A. (1998), '*The Invisible Computer: Why good products can fail, the personal computer is so complex, and information appliances are the solution*', Cambridge: MIT Press.

Poltrock, S. E. and J. Grudin. (1995). 'Organisational Obstacles to Interface Design and Development: Two Participant User Studies'. *In Human Computer Interface Design, (M. Rudisill, C. Lewis, P. B. Polson, T. D. McKay, Eds.z), (SF, Calif.: Morgan Kaufman*), pp. 303-337.

Resnick, P (1992) 'Skip and Scan: Cleaning up Telephone Interfaces' *In Proc. Of SIGCHI'92, ACM Press, pp 419-426.*

Roast, C. and P. Siddiqi (1997) Using the Template Model to Analyse Directory Visualisation' (1997) *Interacting with Computers, Vol. 9(2), Elsevier: Holland, pp-172.*

Williams, David M. L. and Christine Cheepen (1998) 'Just Speak Naturally: Design for Naturalness in Automated Spoken Dialogues'. *Proceedings of CHI 98, ACM Press*, pp 243-244.

Yankelovich, N (1997) 'Using natural language dialogues as the basis for speech interface design', in Luperfoy, S. (ed), *Automated Spoken Dialogues*, MIT Press

Yankelovich, N., (1995) 'Designing speech acts: Issues in speech user interfaces', in *Proceedings of SIGCHI '95*, ACM Press.

Zoltan-Ford, E., (1991) ‘How to get people to say and type what computers can understand’, *Int. Journal of Man-Machine Studies (34),* Academic Press Ltd, 527-547.

Appendix 1: Human Factors Design Checklists

The following checklists are meant to provide the developer a quick overview of important human factors issues both with respect to system or component characteristics and as regards the development and evaluation process. The checklists are based on the discussions in the previous sections.

9.1 Project Grid Checklist

Speech recognition

What kind of input does the system understand (connected words such as connected digits, isolated words, continuously spoken utterances)?

Is there a limit on the length of an utterance?

Is grunt detect / silence detect available?

Is prosodic information interpreted?

Is barge-in supported?

Are confidence levels supported?

How were grammar and vocabulary defined (developer defined, based on empirical user data)?

Natural Language Processing (NLP)

How were grammar and vocabulary defined (developer defined, based on empirical user data)

Dialogue management

Is the dialogue system-directed, user-directed or mixed?

Does the system use an explicit prompt structure (i.e. menu structure)?

Is a command structure supported (e.g. flat hierarchy)?

Is a form filling/frame-based dialogue structure used?

Does the system use a transactional mode or an interactional mode?

Does the dialogue explicitly cater for novices and experts?

Are meta-communication facilities available?

Is a help facility available?

What kind of help facilities are used (single shot, incremental, context dependent)?

Is a human operator backup provided?

Are escape routes out of the dialogue available?

Are explicit confirmations used?

User Model

Are user preferences reflected in the dialogue?

Is a user profile kept?

Domain description

Was this studied before system design?

Were users involved in the description?

Is information provided from a database?
Language generation Are prompts worded such that it is possible to interrupt them before everything is said (i.e. semantically valid cut-off points)? Is a curt/denatured style used?
Speech output What type of output is used (pre-recorded or synthesised speech)? Was any evaluation of speech output quality carried out? Is there use of programmable intonation patterns in system output?
Multimodal Input/Output Are other input modalities than speech supported (e.g. gesture or DTMF)? Does the system support a mix of inputs (e.g. speech and gesture concurrently)? Are other output modalities than speech supported (e.g. graphics or text)? Does the system support a mix of outputs (e.g. speech and text concurrently)? Are non-verbal sounds (e.g. music, background noise, auditory icons) used by the system? Does the system make use of engineered sounds (e.g. earcons)? Does the system make use of a beep to prompt users?
Users For which group(s) does the system provide an interface (agents, caller/user, system administrators)? Is the system designed for naive / novice /expert users? What frequencies of use are mainly supported by the system (often, occasional)? Does this apply to the whole application or only to certain functions? Is knowledge of SLDSs required? Is knowledge of previous service (e.g. agent based system) required? Do first time callers receive guidance from the system? Is knowledge of the application domain required?
Language/culture Does the system cater for different languages? Are different dialects supported? Is the dialogue dependent on the user's gender? Were cultural dependencies identified?
Access and use location How is the system accessed (telephone-based, web-based, on local computer)? Where is the system to be called from/used (home, work, mobile, e.g. car, train)? Are the users subscribers? Are the users paying premium rates for the service?

9.2 Project Life-Cycle Checklist

Overall goals

Are human factors issues considered as part of the overall design goals?

Constraints

Does the customer impose any human factors constraints?

Do the designers impose any human factors constraints on the design which were not dictated from elsewhere?

Are there other (which?) human factors constraints on the development and evaluation process?

Documentation

Are human factors issues documented in the requirements specification?

Are human factors issues documented in the design specification?

Are human factors issues documented in the development process representation?

Are user studies formally documented?

Is evaluation documented?

Criteria

Are real user needs considered?

Is a realistic vocabulary definition done?

Is a core task description done?

Is an effort made to bring functionality criteria and usability criteria in agreement?

Has their feasibility been evaluated early in the process?

Organisational aspects

Are any organisational aspects taken into account?

Are organisational effects of the system considered?

Has a working environment description, e.g. ambient noise etc. been made?

Stakeholders

Is a customer involved?

Are representative users defined?

Are real users involved throughout the process?

Is user experience considered?

Are user backgrounds and background knowledge considered?

Are language/dialectical issues considered?

Do the developers have good mastery of the human factors aspect?

Do the developers spend much time on human factors?

Evaluation

Which evaluation criteria will be used (see the set of criteria provided in Section 5)?

When is evaluation done (during requirements capture, in iterations after implementation, once on implementation)?

Who with (in-house, friendly client, cold client)?

How many users are involved?

Which types of evaluation are used (performance, diagnostic, adequacy, quantitative, qualitative, objective, subjective)?

Which types of tests are used (controlled, field, acceptance)?

Are walkthrough techniques used?

Are mock-ups used?

Is Wizard of Oz (WOZ) used?

Are subjects aware there is a wizard?

Are WOZ recordings used as speech corpus?

Are WOZ recordings used to refine dialogue design?

Do evaluations involve subjects using scenarios?

Are the experimenters aware that scenario wording may influence user utterance wording?

Appendix 2: Draft Template for Evaluating Aspects of SLDSs

In the following we present a modified version of the template in (Bernsen, 1999). It has been adapted to fit the needs of Human Factors. The template is intended to be a generic tool to which corresponds an “empty” template version which must be filled in for each property to be evaluated.

A. What is being evaluated (property)

This entry describes the *property or properties* of an SLDS or component that is being evaluated, such as speech recognition error rate. In some cases, an evaluation criterion refers to a *generic property* which covers several different *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. When dealing with generic properties, the evaluators using the template will have to do the appropriate additional specifications of the specific properties which they will be evaluating. The filled evaluation template should mark whether the criterion concerns a specific or generic property.

B. System part evaluated

This entry describes which component(s) of an SLDS are being evaluated, if any. This could be, e.g., the speech generation component or the system as a whole.

C. Type of evaluation

This entry describes the *type* of evaluation, i.e. whether evaluation is quantitative, qualitative or subjective and whether or not evaluation is comparative. Some evaluation criteria are comparative in nature. Many others can in principle be used for comparative evaluation. It is, of course, satisfying to obtain a quantitative score from the evaluation which can be used to measure progress, and which may even be objectively compared to scores obtained from evaluation of other SLDSs. However, many important evaluation issues relating to SLDSs cannot be subjected to quantification. Note that a particular property under evaluation may be subjected to several different types of evaluation.

Terminology

Quantitative evaluation consists in counting something and producing an independently meaningful number, percentage etc. It should be noted that, even if quantitative measures may make little sense in absolute terms, i.e. as independently meaningful numbers or scores, quantitative measures can be useful for progress evaluation in which improvements are being measured against, e.g., a test suite. However, we would argue that quantitative progress evaluation is not considered ”real” quantitative evaluation as long as progress is not being measured against an independently meaningful quantitative standard or target. Independently meaningful scores are not only very important for purposes of comparative evaluation of systems and components, they are also difficult to achieve. For instance, many published speech recogniser recognition success rates suffer from under-specification 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. The standards to apply may derive from the literature, from experience or from expert consultants.

Quantitative and qualitative evaluation are both *objective evaluation*.

Subjective evaluation consists in judging some property of an SLDS or, less frequently, component by reference to users' opinions.

Comparative evaluation consists in comparing quantitative, qualitative or subjective evaluations for different SLDSs and components. Comparative evaluation is often done internally in a development process in order to measure progress (progress evaluation). In most cases, this does not produce independently meaningful scores which can be used in comparisons with other SLDSs or components. The individual filled templates generally ignore such *internal comparative* evaluation. An equally important but much more difficult form of comparative evaluation is comparison between different SLDSs or components. The general problem with this *external comparative* evaluation of SLDSs and components is that it can be difficult to ensure evaluation under strictly identical conditions, such as same task, same test suite, same-sized user population etc. As a rule, the easier it is to ensure strictly identical conditions, the more specific is the property being evaluated. However, customers and end-users tend to be more interested in global evaluations that take into account many different properties, asking: which SLDS or component among several is globally the best one? Such evaluations are at best qualitative and often include subjective elements.

D. Method(s) of evaluation

This entry describes the methods of evaluation that 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 E below).

Terminology

Design analysis consists in using experience and common sense, thinking hard when exploring the design space during the specification and design phases, doing walkthroughs of models, using mock-ups, comparing with similar systems, browsing the literature, applying existing theory, guidelines and design support tools, 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 considered [Bernsen et al. 1998a]. Evaluation also consists in checking whether goals and constraints are sound, non-contradictory and feasible given the resources available. Note that design analysis can be performed at any time during the life-cycle, not only during the early design phase. For instance, a customer considering alternative offers may want to analyse the requirement specifications and design specifications of the products on offer.

Wizard of Oz data analysis consists in analysing problems posed by phenomena observed in data from simulated user-system interactions. The simulations are performed by one or several humans and address the non-implemented 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 recogniser 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 of analysing the generated data. The perception of the SLDS or component by the users involved in the simulations can be investigated through methods such as questionnaires and interviews.

During debugging of the implemented SLDS or component, two typical types of test are glassbox tests and blackbox tests.

A *glassbox test* is 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 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 if 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.

Test suites are useful for evaluating one or several sub-components independently of the rest of the system. Use of test suites for component evaluation should always be accompanied by rigorous and explicit consideration of the match between the test-suite evaluation conditions and the actual operating conditions for the component in the integrated system. Any mismatch, such as lack of representativeness of the test suite data or of the acoustic signal conditions, may render the test suite evaluation results irrelevant to judging the appropriateness of the component for the task it is to perform in the integrated system.

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 which they have to perform, or in *field studies* where the SLDS or component is being exposed to uncontrolled user interaction. User-system interaction data is useful or even necessary in many cases, when too little is known in advance about the phenomena that will be present in the deployed application. This data, if comprehensive, has high reliability because of deriving from a test corpus of sufficient size and realism wrt. task and user behaviour. Unfortunately, the data cannot be obtained until late in the development of the system. User-system interaction data analysis, if performed extensively rather than cursorily, is costly. This kind of analysis can be partly replaced by Wizard of Oz data analysis which is costly as well but which happens early enough in the life-cycle to enable prevention of gross errors. Since there is significant cost in both cases, cost which is only offset by corresponding risks, this is where (early) design support tools are most desirable.

No standards exist for which questions to ask in *questionnaires and interviews*. No standards exist on how to interpret the results of questionnaires and interviews. Still, these methods can give crucial insights into the users' perception of the system. For *questionnaires,* a standard procedure is to ask users to express their subjective perceptions of the SLDS as a series of properties on a five-point scale. Questionnaires should contain a "free-style comments" section. *Post-trial interviews* are useful for capturing user observations which might otherwise have been missed and which might have implications for virtually any kind of system deficiency.

E. Symptoms to look for

This entry describes the symptoms the evaluator should look for in the data. These could be, e.g., lack of understanding by the system, apparently irrelevant system responses, or user complaints in a questionnaire.

F. Life-cycle phase(s)

This entry describes the life-cycle phases in which evaluation of the property in question should be performed. In general, the earlier evaluation can start, the better. Distinction is made between early design, simulation, implementation, field evaluation, final evaluation, maintenance and porting.

Terminology

Early design including requirements and design specification. Pointedly expressed, 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 and 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 explains the importance of applied theory, guidelines and tools in support of early design.

Simulation and implementation. This are the life-cycle phases in which modules, such as the dialogue manager and its sub-modules, should be severely tested. To begin with (part of) the SLDS 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 can 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 SLDS or component to uncontrolled interaction with users. Field evaluation may precede the final acceptance test.

Final evaluation may consist in an acceptance test, i.e. a more or less formal and controlled evaluation experiment which should decide if the system, such as an SLDS, meets the evaluation criteria specified as part of the requirements specification. What is primarily being evaluated is the behaviour of the system as a whole. In addition to controlled experiments, final evaluation may include design analysis and blackbox tests. The evaluation methods used during final evaluation may also be used for *customer evaluation* in which a potential customer wants to understand the positive and negative sides of an SLDS or component.

Maintenance deals with updating the SLDS or component in various ways, such as updating the database linked to the dialogue manager.

Modification deals with re-using the SLDS or component for new purposes. This includes localisation, customisation, additions and other kinds of changes.

G. Importance of evaluation

This entry comments on the importance of evaluating a certain property. 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?

- if the system or component has 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. Stating those reasons is important to understanding the grading proposed. For instance, it may be quite crucial to get some property right even if that property, such as speech acts identification, is relevant only to few current systems at this point.

H. Difficulty of evaluation

This entry comments on the difficulties 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;

I. Cost of evaluation

This entry comments on the costs involved in performing the evaluation.

- evaluation is more or less *costly* to perform in terms of time, manpower, or skilled labour;

- difficult evaluation may be relatively uncostly, for instance if done by a consultant; easy evaluation can be costly, for instance because of the volume of data involved.

- Wizard of Oz simulations, field studies and data analysis are costly.