Chapter 12
Methodology II: Cognitive Dimensions and the Gulfs

Abstract This chapter introduces two useful high-level approaches that summarize a wide range of aspects of users: how users interact with artifacts and how users perform tasks. A type of analysis, Cognitive Dimension/dimensions, provides a way to represent common and important aspects of interface design. The dimensions are used to describe and then evaluate interfaces. Norman has a similar description of user behavior based on how hard it is to understand the state of the interface (the Gulf of Understanding) and how hard it is to perform an action in an interface (the Gulf of Execution). Both approaches are useful techniques for thinking about, planning, and performing interface design.

12.1 Introduction

So far we have presented several methods and techniques for analyzing Human-System Interaction, including task analysis as a general tool, and GOMS and the KLM as more detailed formal approaches for use prior to user testing. Most of the methods we have presented are applied at a low level of abstraction. In contrast, the Cognitive Dimensions (of Notations), usually just referred to as Cognitive Dimensions or CDs (e.g., Blackwell and Green 2003; Green 1989), was developed as a mechanism for discussing and analyzing systems at a higher level of abstraction.

The CDs are based on the assumption that it is useful to have a language for discussing design. One way of trying to understand design is to describe the object that is being designed, and the trade-offs that the designer faces using a small number of fundamental dimensions. These dimensions can then provide a common language (sometimes called an ontology) for discussing usability problems, as well as a means of comparing aspects of interface and system design, and for comparing whole designs. Thus, the CDs can be seen as a way of providing a common ontology for naming aspects of design as well as for naming design trade-offs.

F. E. Ritter et al., *Foundations for Designing User-Centered Systems*,
DOI: 10.1007/978-1-4471-5134-0_12,

The goal of the CDs is to provide a fairly small, representative set of labeled dimensions that describe critical ways in which interfaces, systems, and environments can vary from the perspective of usability. Once the dimensions have been labeled the designers can more easily discuss alternative designs.

In this chapter we will outline some of the dimensions and illustrate them with examples. The complete set of dimensions, however, does not yet exist—this is still a research issue and the set of dimensions for a given design is probably tied to the context of the design. Research in this area is proceeding in two directions: trying to find ways of formalizing and measuring the dimensions currently proposed, and trying to explore the completeness and redundancy of the current set of dimensions.

We will also describe an alternative viewpoint on dimensions, based on the work of Norman (1988, 2013). Norman's dimensions describe how users interact with devices, focusing on the relationship between actions and intentions, and between the results of actions and the way they are interpreted.

12.2 The Cognitive Dimensions

Table 12.1 notes the (current) set of 14 Cognitive Dimensions. Research into the CDs is ongoing, so the list should not be regarded as finalized or complete. The CDs can be extended to make them more appropriate to your own situation.

Below we describe the first five CDs in more detail. Our main purpose here is to illustrate what the CDs are, and how they can be used.

12.2.1 Hidden Dependencies

Dependencies are the number and direction of relationships between objects. For example, spreadsheets show you formulae in one direction only, that is, which cells are used to compute the value in a cell, but not which cells use a given cell's value; similarly, variable declarations in programming languages like Java or Pascal show you which variable type a structure is made from, but not the other way around; and the Microsoft Word Styles dialogue box (Fig. 12.1) shows you the parent of a style but not its children.

In all of these examples there are two hidden (or invisible) types of dependency: ancestors and children. Only the parents are visible, and finding the ancestors involves following a chain of parents. Working in the opposite direction, finding the children involves checking every entity for its parent, which creates a high memory and work load. Another example of the problem of hidden dependencies occurs in variable initialization and use: in many procedural and production system languages you need to use a combination of tools and deep knowledge to find out what the dependencies are.

Table 12.1 The Cognitive Dimensions

1. Hidden dependencies: how visible the relationships between components are
2. Viscosity: how easy it is to change objects in the interface
3. Role-expressiveness: how clear the mapping of the objects are to their functions
4. Premature commitment: how soon the user has to decide something
5. Hard mental operations: how hard are the mental operations to use the interface
6. Secondary notation: the ability to add extra semantics
7. Abstraction: how abstract the operations and system are
8. Error-proneness susceptibility: how easy it is to err
9. Consistency: how uniform the system is (in various ways, including action mapping)
10. Visibility: whether required information is accessible without work by the user
11. Progressive evaluation: whether you can stop in the middle of creating some notation and check what you have done so far
12. Provisionality: whether you can sketch out ideas without being too exact
13. Diffuseness: how verbose the language is
14. Closeness of mapping: how close the representation in the interface (also called notation) is to the end results being described

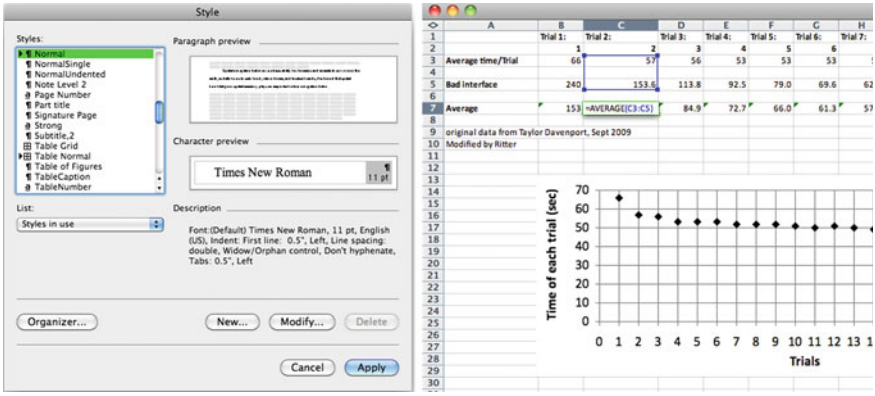

Fig. 12.1 The Word Style editor (*left*) shows the parent of the ref style, but cannot show the children of the Normal style. The spreadsheet (*right*) shows what cells the formula is dependent on (and can show those that appear as the line on the graph), but does not show where a given cell is used

The implications of considering hidden dependencies are that all dependencies that may be relevant to the user's tasks should be represented. Or, as a minimum, tools should be provided which enable them to be represented. This is consistent with the notion we saw earlier in the book of recognition being easier than recall. For example, spreadsheets would be easier to use (for certain tasks anyway) if they could show forward relationships (e.g., this cell's value is used in all these other places). Recognizing this fact (the need for visibility) is quite separate from designing a solution to the problem—one option would be to use color coding (e.g., child cells could be shown in the same color; ancestor cells could be shown

in ever paler shades of the same color). Some spreadsheets now provide the capability to show child cells, thus making the dependencies less hidden.

A much richer form of hidden dependency occurs in many modern operating systems, where several files are both generated and used. Applications other than those that created them may be dependent upon these files, such as a pictorial figure in a spreadsheet file, a graphics file used in a report, or a preferences file stored somewhere away from its application and labeled with the name of the company or user rather than the application. These dependencies are not visible—deleting files from system and user preferences directories therefore becomes a hazardous task. The corollary is that people's file stores get filled up with old, unused files and you cannot easily tell whether or not they are used. The normal solution, albeit one that strictly ignores the problem rather than solves it, is to never delete any files from these directories. The reason this works as a solution is that storage space is so cheap!

12.2.2 Viscosity

A *viscous* system is one that is resistant to change—even small changes can require substantial effort. A classic example would be a word-processed document in which the numbers used in the figure captions, such as "Fig. 12.1," have been typed explicitly by hand (rather than using the word processor's built in numbering facilities). If you decide to introduce another figure before this in the document, then all the existing figure numbers need to be changed manually too. Whilst this may not be too difficult for the figures themselves, ensuring that all the references to the figures in the text are updated to match the updated numbering can require significant effort. If you introduce several figures in this way, one at a time, you may decide to skip doing the updating of the numbering every time, and in the end may forget to do it at all. TeX and LaTeX avoid this problem by doing it automatically, and Word also provides tools for doing this, although these tools are somewhat viscous and error prone.

In some circumstances viscosity can be beneficial for users. If things are hard to change this encourages reflective action and explicit learning. Learning in some puzzles (noted in Chap. 6) is more explicit if the user has to work harder to make moves in the puzzle. Deleting files is also made somewhat more viscous by many operating system interfaces ("Are you sure you want to delete this file?"). In contrast, if things are easy to change this encourages tinkering, which leads to implicit learning. When it is very easy to make small changes this can often lead to many small, unnecessary changes being made.

On the other hand, visualprogramming languages, one of the original analyses that led to the CD approach, can have changes that are difficult to implement and change. For example, it is most productive in most cases to vary fonts uniformly and to have them vary by several points. It is not as productive to have them vary

by only one point. If you want to modify the fonts in a PowerPoint presentation by one point, then you have to use more detailed tools and it takes longer. You can do it, but it is more viscous. As another example, iTunes makes it relatively easy to add a new playlist or to modify a playlist. It makes it more difficult (viscous) to erase songs.

Two types of viscosity can be differentiated:

- Repetitive viscosity—where what appears to be a single goal has to be carried out as lots of small, repetitive actions. For example, if you want to change in a document every number followed by a tab into the same numbers followed by a space, this has to be done by hand in Word because, while the number can be found, it cannot be included in the replace, thus increasing the viscosity of editing Word documents. Similar effects can be found in many systems when renaming files within a graphical user interface.
- Knock-on viscosity—where what appears to be a single goal requires several more small changes to restore consistency (e.g., adding a sentence at the beginning of a document and having to redo all the work involved in ensuring that appropriate page breaks occur).

Solutions to the problems of inappropriate amounts of viscosity can involve either redesigning the system or providing tools to help manage the difficulties. The latter is more likely to be useful where viscosity may be desirable, as a way of promoting reflective action and encouraging learning. In some cases there may be a trade-off, however, and making some actions less viscous may lead to others becoming more viscous. It is also worth noting that it can be useful to deliberately make some actions viscous, e.g., if there are safety critical elements, or an action may be dangerous if carried out incorrectly, or they are expensive in time or money. On the other hand, simple, routine actions should normally be made as non-viscous as possible.

12.2.3 Role-Expressiveness

Role-expressiveness describes the extent to which a system reveals the goals of the system's author/designer to the reader/user. It should be easy, for example, for a reader to see what each component in a program does, and how each component relates to the whole. This need to infer the other agent's goals and plans is important, as we noted earlier in Chap. 7, with particular reference to how Grice's maxims describe successful communication.

One obvious example of role-expressiveness is that the buttons on an interface should be clearly recognizable as buttons that the user can press. In some interfaces, however, role-expressiveness is poor because interface objects that are clickable are not easily recognizable as being clickable—"But how was I meant to

know that it would do something if I clicked there?" This effect is particularly found on those web pages where banners and logos are designed as clickable images, but users more commonly (and more naturally based on their experience) perceive them as just being graphical images.

A piece of program code with well-chosen variable names can be very role-expressive. The goals of the programmer in each statement can be made readily apparent to the reader if the functions and variables are given names that clearly indicate their purpose.[1]

The Microsoft Excel spreadsheet application also supports role expressiveness. When you add a formula to a particular cell, for example, it color codes the cells that are referenced by that formula. Note that the color coding only happens as the formula is created, and disappears when you calculate the result of evaluating that formula.

Classic problems of role-expressiveness occur where two similar looking features achieve different functions or where two different looking functions achieve similar effects. That is, where the role-expressiveness for two similar looking objects is different, or where the function of two different looking objects appears the same. For example, the *Border and shadings* functions in some older versions of Microsoft Word can be accessed using the top level menu but also appear in three other places in the interface, with different sets of functions available to the user in each place with differing effects. This can confuse users who fail to distinguish between paragraph borders and table cell borders because Word displays them in apparently unpredictable ways.

There may sometimes be a conflict between role-expressiveness and consistency. Designing a system to be role-expressive will usually mean using a richer vocabulary with less (apparent) uniformity. A good way to resolve this conflict is to carry out a clear and effective analysis of the users' tasks and the importance of learning versus other measures of usability rather than focusing solely on consistency (Grudin 1989). Role expressiveness might also be poor on purpose to encourage exploration. This may occur in games or educational software and hardware (Yeh et al. 2010).

12.2.4 Premature Commitment

The point at which users have to commit to taking a particular action varies across environments and systems. For example, when you go into a café you sometimes find that you have to select which cutlery you need before you have seen what food is on offer that day. Until you know this you will not know whether you will need a

[1] Jonah Gregory (personal communication, 2008) notes that programmer's comments can be helpful in these situations, and we agree. However, comments can also be a nightmare because they often do not get updated when the code changes. So, there are design decisions about when to update and how to update such documentation.

soup spoon or a dessert spoon (or both). Similarly, many database systems require that you plan and commit to particular record structures and size limits before entering any data or actually using the system. Until you know how the system will be used, however, it is often difficult to finalize these decisions.

These limitations are examples of *Premature Commitment*. Computer programming environments often require premature commitment such as declaration of variables before working out how they will be used, and developing the system in sequential order. Like many of the CDs, premature commitment will interact with other dimensions. Premature commitment can contribute to viscosity, for example, because the effect of the commitment is to make future changes very hard to make.

The solution to the problem of premature commitment is usually to allow (and provide support for) users to perform tasks in many orders (e.g., outlining tools in many word processors allow the user to develop the outline and write the text in any order). This support comes at a cost, however. Allowing users more freedom often makes the system more complex to design and to build, and may contribute to errors in program code. Paying attention to this trade-off, however, is completely consistent with the Risk-Driven Incremental Commitment Model that we will discuss in Chap. 14.

12.2.5 Hard Mental Operations

The final CD that we will examine here is the number of *hard mental operations* involved, which will vary across systems. In GOMS, KLM modeling, and several other task analysis methods, the mental operations are all assumed to be equivalent, for convenience's sake. Psychology and human factors work show us that some kinds of problems (and operations) are substantially harder for users to perform, and that users thus prefer easier mental operations (although harder does not necessarily mean slower).

The issue of hard mental operations can be illustrated by the isomorph of the Towers of Hanoi problem, in which monsters manipulate balls according to a formal monster protocol for swapping balls. The subjects in Kotovsky and Simon's (1990) study had monsters either follow a complex protocol to swap balls around based on which size monster could hold which size ball, or a protocol that required monsters to change size to swap balls around. Subjects found the latter protocol to be much more difficult, possibly because we tend to think of size as a relatively constant aspect of an object, so having to mentally change the size of an object is more strenuous and error prone than applying simple rules to behavior. Similar effects have also been found in mentally rotating objects, with larger objects being rotated more slowly.

As another example, multiple negative facts can be very hard to disentangle (e.g., the packet that was not lost, was not put out on the Ethernet). The concepts of pointers and indirection in C and other programming languages also prove very

difficult—most people can usually handle one level of indirection, but multiple levels of pointers tend to be more difficult to follow. Another type of hard mental operation is boundary problems, for example, counting fence posts—If you have a ten-foot fence to put up, and fence pieces are one foot long, how many posts do you require? In this situation, the boundaries of an object have to be explicitly and carefully represented and manipulated. If you thought there should be 10 posts, you have left out the first post, because there needs to be 11 (you can see this much more easily if you actually draw the fence).

An important thing to remember about these kinds of hard mental operations is that they can be easy to implement computationally, but can be especially troublesome for users. Again, these problems can be solved at several levels, including either by avoiding the problem by understanding the relative difficulty of operations, or by providing tools to assist in these operations (e.g., graphic displays of indirection in C programs) and providing representations designed to be shared between types of system implementers.

12.3 Turning Cognitive Dimensions into a Methodology

The CDs can be used as a method for informing design during the early or middle stages of development (e.g., Cohen et al. 2012). The designer can ask questions based on the appropriate dimensions for the system, such as:

1. Are there hidden dependencies and one-way links in the structure?
2. Is the system viscous, or can users and designers reconstruct structures easily and fluidly?
3. Is the order of generation and action the same as the order in which users will want to do things and have the appropriate information? Are there multiple, appropriate orders of behavior and strategies supported?
4. Is the system role-expressive? Are the functions of the components obvious and discriminable?
5. Does the system require any hard mental operations?
6. Are there conceptual gaps between the user and the system? (Blandford et al. 2008)

The answers to these questions are shared among the design team. They can then be used to moderate the design of the system, or may lead to other design activities such as studying the relative costs of mental operations or other tasks that help reduce the risks associated with the design.

Designers might wish to order the dimensions they use, and they may also wish to add other dimensions. A new dimension could be added, for example, to consider the role of cognitive load (Sweller 1988) describing how much the user has to know, do, and learn at any given point in a task.

The CDs have been used to develop extensions to the Microsoft Excel spreadsheet that integrate user defined functions into the worksheet grid (Peyton

Jones et al. 2003). This was done by initially focusing on the cognitive requirements of the user, using the CDs and the Attention Investment model of abstraction use (Blackwell 2002). The Cognitive Dimensions approach has also been used to understand issues that arise for programmers when solving complex systems integration problems and with contemporary, *mash-up*programming practices with APIs (application programming interfaces) (see Jones et al. 2010 for an example).

12.4 What is Omitted by the Cognitive Dimensions?

The CDs do have some limitations. This approach does not offer a complete way to create an interface design, and it may not always be applicable. Those aspects of a system that are of particular interest may not be covered by the dimensions noted, for example.

Even for the existing CDs, there are areas where they could be expanded. The current list includes error proneness, for example, but does not directly address the quality of feedback and support for error recovery. The CDs also focus on usability issues, and do not currently address issues related to user acceptability—does the system fit in with the way users do the task or want to do the task, and so on.

The CDs are an attempt to provide a way for designers to discuss issues related to usability. If we had a rich enough vocabulary to adequately describe the sources of poor usability, then the problems could be solved simply by reducing or eliminating each of these sources. The real benefit of the CDs is as a tool that can highlight the trade-offs inherent in interface design in a way that can be easily understood by designers.

As their name suggests, the CDs focus on the cognitive aspects of interfaces. They do not address design trade-offs related to the other aspects of users that we have introduced—anthropometric, behavioral, and social aspects—in any great depth. The CDs approach, however, suggests that a similar approach to understanding design trade-offs could be used for these other aspects of usability. The physical dimensions of a device will typically be related to the features that can be provided, for example, iPods versus iPads. Similarly, other aspects can be put into this framework. For example, social aspects could be included such as how the size of a social network will influence the usability of devices designed for communicating with others.

12.5 Norman's Seven Stages of Action

In his book *The Design of Everyday Things*, Norman (1988, 2013) describes the process of how users interact with systems to achieve their goals as a series of activities (see Table 12.2). This process is an approximate theory of action: all of

Table 12.2 Stages of user activities

• Establish the goal
• Form the intention to take some action
• Specify the action sequence
• Execute the action
• Perceive the system state
• Interpret the system state
• Evaluate the system state with respect to the goals and intentions

the stages may not always be used, and they may not always be applied in the same order. The stages are done in a cycle (thus, the last stage, *Evaluate*, also occurs before the first stage, *Establish the goal*). The process does, however, capture the essential aspects of how users perform tasks, and is therefore useful for analyzing and guiding the design of systems. The process should be seen as cyclic rather than a single shot linear sequence of activities. Activities may also overlap, and there can be feedback loops between activities.

There may be not be a direct correspondence between the user's psychologically expressed goals and the physical controls and variables that have to be used to carry out the task to achieve those goals. In many cases the goals and intentions are highly context dependent, and often opportunistic rather than planned. The goals and intentions may also be rather vague or ill-defined. Generally the user will set up some goal that they want to accomplish, and then formulate an intention to carry out some actions to ensure that the goal is achieved. Goals and intentions are psychological concepts—they exist in the mind of the user and relate directly to the user's needs and concerns. In most cases, however, the actions have to be performed on a physical artifact or system, manipulating some physical mechanisms to produce changes in the physical variables and the state of the system or environment. The user must, therefore, turn psychological intentions into actions that can be physically carried out upon the system or environment. The user then has to look for the effects of those actions, and interpret those effects in terms of the psychological goals to decide what to do next.

You do not want your users to have to resort to effortful problem solving just to interpret the effects of any actions. If they do, this usually means that they have not found the interface easy to use. This situation should normally only occur when something goes wrong with the system, or where the purpose of the interface is to deliberately force the user to resort to problem solving, such as in a game. Providing some level of built in redundancy will help users when they have to perform problem solving. You could provide them with multiple email accounts, for example, in case one is down, or provide multiple ways to get email, e.g., POP and webmail; similarly, you could provide multiple ways to print with multiple printers. These are usefully redundant rather than just duplicates.

When you design systems you should think about how to support users in ways that help them to achieve their goals, and reduce the need for them to resort to problem solving. You can provide users with some means to access commands, for example, using keystrokes, menus, buttons, and so on, as a way of helping them

perform their tasks to achieve their goals. You should also try to anticipate potential problems, and understand how your users will use problem solving to resolve them. By doing so you should be able to identify redundancies that you can build into the system to allow (and help) users to work around the problem until it can be fixed.

12.5.1 The Gulfs of Evaluation and Execution

In Norman's seven stages of action he explicitly talks about the need to map between psychological and physical concepts. Often there is not a direct correspondence between the two. If we think about behavior in terms of Neisser's (1976) perceive-decide-act cycle we can identify two areas where the mapping occurs.

When the user perceives the state of the system (or more generally, the environment) this will be in terms of physical concepts (usually variables and values) that the user will have to translate into a form that is compatible with their mental model of how the system operates. Norman describes the gap between the physical concepts and the psychological concepts as the *Gulf of Evaluation* (see Fig. 12.2). The user will normally have to do some (cognitive) work to bridge this Gulf. If the user cannot interpret the effects in terms of the original goals and intentions, this may mean that they have to resort to problem solving to determine whether the effects of their actions indicate that the goal has been achieved.

When the user formulates a goal, this usually leads to the intention to perform some actions to attain that goal. The user then has to translate these psychological concepts into physical concepts, which are usually actions that can be executed on the system (or, more generally, the environment). Norman describes the gap between the (psychological) goals and intentions and the physical actions as the *Gulf of Execution* (see Fig. 12.2). If the user cannot readily work out how to translate the goals and intentions into the available actions that can be performed on the system, they may end up not knowing what to do, or how to do it. In such cases they will often have to resort to problem solving to determine what actions to take and how to execute them.

When designing systems you should normally try to minimize the size of the Gulfs of execution and evaluation. Often it may not be possible to eliminate them completely, but you should try to make it obvious what the user needs to do to bridge the Gulfs.

12.5.2 The Gulfs in Practice

A couple of examples should help to illustrate the two Gulfs more clearly, and show how serious the effect of large Gulfs can be. In the Kegworth air crash (see Appendix) and in many other air disasters a large Gulf of Evaluation can be identified. The true state of the aircraft was not very clear from the instruments in the cockpit, and there

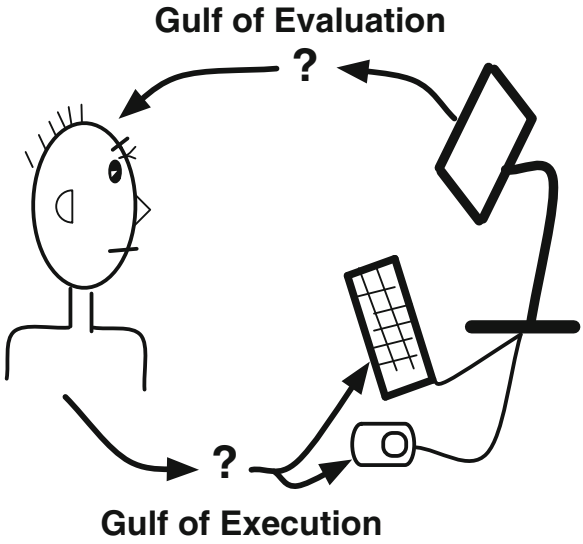

Fig. 12.2 Norman's Gulf of Evaluation and Gulf of Execution

was little directly observable feedback about the effects of the pilots' actions. There were also some apparent discrepancies between the pilots' mental model of the state of the aircraft and the state of the physical aircraft, and the instrumentation in the cockpit did not make it easy to identify and rectify these discrepancies. In the Kegworth disaster the air traffic control operatorsasked a lot of questions. These questions may also have contributed to the Gulf of Evaluation for the pilots because they would have taken time away from understanding their situation.

The situation in the Kegworth disaster was further exacerbated by coincidental effects that led the pilots to falsely believe that they had attained some goals (Besnard et al. 2004). There was an identifiable Gulf of Execution in that the actions that the pilots took did not have the effects they thought they would. While they had removed the vibration by shutting down one of the engines, they had not shut down the failing engine.

As you look at other interfaces, you will encounter further examples where details and feedback on the state of the system can be difficult to interpret, and where it can be difficult to work out what actions are available and how to execute them. These sorts of problems provide an indication of where the usability of the system can be increased by improving the interface.

12.6 Implications of the Gulfs for Design

Norman uses the Gulfs to highlight the central importance of visibility in design. Good design involves making sure that information that is crucial to task evaluation and performance are made clearly visible to the user. In this way you can

reduce the size of the Gulf of Execution and the Gulf of Evaluation. Table 12.3 provides some general guidance on how you can try to narrow the Gulfs during design.

Visibility is not an adequate guideline on its own, however, and making too much information visible can be a bad thing. Norman uses the example of a washer/drier that had multiple controls and several symbols displayed on the front of it. There was no easy way to discern what each control did and what all the symbols meant without looking at the instructions or undergoing extensive training. (This example assumes that the designer was not trying to drive the user to the manual to learn these symbols, a type of controlled viscosity.)

There is often a design trade-off between presenting too much information and overloading controls to allow them to perform several different actions using modes. In moded systems, the way that you interpret the displays and feedback depends on which mode the system is currently in. This can be problematic, particularly if it is not immediately obvious which mode the system is in.

The visibility guideline, like most guidelines, needs to be adapted to the user's particular situation. You should think of it in terms of:

> "Visibility of the appropriate information for carrying out relevant tasks," or
>
> "So viel wie nötig, so wenig wie möglich." → "as much as necessary, as little as possible," is a neat way of phrasing it, taken from a German aphorism.

What counts as appropriate information will vary across tasks, and sometimes across users, and even across contexts of use (e.g., same driver, same task, different road conditions).

When you think about visibility you should make sure that you give appropriate consideration to feedback, consistency, and the user's mental model of the system. These factors can all be used to help reduce the size of the Gulfs. Feedback is obviously important in helping to reduce the Gulf of Evaluation, because it shows the effect of performing a particular action.

Users often rely on consistency within and across systems and applications. They expect interface layouts to be consistent, for example, with particular controls always being located in the same place and always behaving in the same way. This can be particularly important when it comes to new systems, because users will often try to apply their knowledge of other interfaces to the new interface. Consistency is important because it will help the users to help themselves.

We know that users rely on mental models to help them perform tasks. These mental models develop over time, so your design should facilitate the development of appropriate mental models, and support the use of those mental models by making the appropriate information visible to users at the right time and in the right place.

In many cases you want it to be easy to use the system you are designing, so that you can get to the point where people can use it with little or no conscious control. In other words, they will be operating at a point close to Rasmussen's (1983) skill-based level of behavior.In critical systems, however, you often want people to pay close attention to the actions that they are taking, and for behavior to

Table 12.3 Design principles derived from Norman's analysis to make the Gulfs narrower where appropriate

1. Use both the knowledge in the world and the knowledge in the head. Provide information in the environment to help the user determine the system state and to perform actions, such as explicit displays of system state, and affordances on the system controls
2. Simplify the structure of tasks. Require less of the user by automating subtasks, or using displays to describe information without being asked, or provide common actions more directly. However, do not reduce this below their natural level of abstraction
3. Make the relevant objects and feedback on actions visible. Bridge the Gulf of Evaluation. Make the state of the system easier to interpret
4. Make the available actions visible. Bridge the Gulf of Execution. Make the actions the user can (and should) perform easier to see and to do
5. Get the mappings correct from objects to actions. Make the actions that the user can apply natural
6. Exploit the power of constraints, both natural and artificial, to support bridging each Gulf. Make interpretations of the state and of possible actions easier by removing actions that are not possible in the current state and reducing the complexity of the display for objects that are not active or available
7. Design for error. Users will make errors, so you should expect them and be aware of their effects. Where errors cannot be prevented, try to mitigate their effects. Help the user see errors and provide support for correcting them
8. When all else fails, standardize. If the user does not know what to do, allow them to apply their knowledge of existing standards and interfaces

be more deliberate (closer to Rasmussen's rule-based level of behavior). Making interaction harder can be useful for safety critical systems (e.g., nuclear power, aviation, and so on), security critical systems, and more generally as a way of preventing inappropriate, costly, and illegal actions in any interface. If you want to make the Gulfs wider, you should apply the inverse of these principles; some are noted in Table 12.4. Figure 12.3, for example, shows a picture of an electrical outlet designed, one can only presume, to be difficult for travelers to use.

At this point it should be clear to you that you are not simply designing to make the system fit the user in isolation. At the same time, you need to consider how to mold the task environment to fit the user. In other words, you need to fit the environment to the tasks, goals, knowledge, and expectations of the user.

12.7 Limitations of the Gulfs

While Norman's approach has been very useful as a metaphor and approach for improving interface design over the years, it has some limitations that should be noted. The most obvious limitation is that it is intentionally simplistic. The suggestions that it makes are context independent. The sizes of the Gulfs are not measurable, and this makes it difficult to apply this approach to choose between alternative designs.

Table 12.4 Design principles derived from Norman's analysis to make the Gulfs wider where appropriate

1. Hide components: Make things invisible to help make the visible easier to see, and what should be hard to find (because it is uncommon or unhelpful or dangerous), is hard to find
2. Avoid natural mappings for the execution side of the action cycle, so that the relationship of the controls to the things being controlled are inappropriate
3. Make the unavailable action physically impossible or difficult to do
4. Require precise timing and difficult physical manipulation
5. Do not give any feedback
6. Use unnatural mappings for the evaluation side of the action cycle, so that the system state is difficult to interpret

Fig. 12.3 An electrical outlet (*circled*) on a column designed to be difficult to use?

To fully understand the Gulfs, designers need to understand perceptual and cognitive psychology fairly well (which is partly why we have included lots of information on cognition and perception in this book). To apply this approach the designer has to be able to accurately judge the Gulf of Evaluation, which implies understanding how the user perceives and interacts with the display. It also

assumes that the designer either understands motor output fairly well, or that the details of motor output do not matter. When the interface is either simple, or has been poorly designed, it will be relatively easy to make judgments about where it can be improved. For more complex interfaces, where there are realistic design trade-offs that need to be made, it will be more difficult to make these judgments accurately and for a wide range of users.

12.8 Summary

The Cognitive Dimensions and the Gulfs of Evaluation and Execution both provide a less formal and much lighter weight approach than task analyses and user modeling techniques. They provide useful heuristics about interface design, and a way to describe interface trade-offs, but they cannot always be used to compute the quality of alternative designs.

The Cognitive Dimensions provide a way to discuss particular aspects of interface design. They are not a complete method, but do provide a useful way to carry out an early sanity check on a particular interface design. It is useful to have such techniques that can be used early on and with little effort.

The Gulfs of Evaluation and Execution remind us how users interact with a system. Nearly all users have some common tasks related to interaction, including evaluating the system state and executing actions. Bridging these gaps (Gulfs) will mean creating the system to support these two fundamental aspects of task performance. Thinking about the Gulfs can yield practical suggestions for improving designs, such as making things more visible but avoiding clutter.

12.9 Other Resources

There are many resources related to Blackwell and Green's Cognitive Dimensions available at their web site. These include papers, reports, and tools.

http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/

Don Norman's book *The Design of Everyday Things* (originally published as *ThePsychologyof Everyday Things*) includes several examples of the Gulfs of Evaluation and Execution arising out of poor design. These include a fridge freezer, doors on buildings, and a slide projector where how to evaluate or how to use the devices were not clear. He also discusses how the designs could be improved to reduce or eliminate the Gulfs:

Norman, D. A. (1988). *The psychology of everyday things*. NY: Basic Books. (2013 ed., *The design of everyday things*).

Norman's (2009) text is also worth reading. This book covers technologies like self driving cars, and applies the basic philosophy laid out in the earlier *Design of Everyday Things*:

Norman, Donald A. *The design of future things*. Basic Books, 2009.

12.10 Exercises

12.1 Consider a smartphone, either a specific one or a composite one. Attempt to come up with a set of trade-offs based on Cognitive Dimensions. Describe how to choose a reasonable point in these trade-offs.

12.2 Choose an existing interface, find five similar interfaces, and note how they are related as different positions on Cognitive Dimensions. For example, there are puzzle interfaces where you directly manipulate pieces and others where you type in the piece number. These differ in viscosity and hard mental operations. The more viscous interface encourages more hard mental operations and learning. Find similar trade-offs within a single type of interface.

12.3 Look at your work desk, and consider where you can note three trade-offs or effects in its design with respect to the Cognitive Dimensions. For example, some desks have printouts pasted to walls near the desk. These can be seen to reduce the viscosity of the information on them.

12.3 Discuss how an online training system and a safety critical system can encourage different behaviors (e.g., learning, avoiding errors) by using Cognitive Dimensions.

12.5 Consider ways that the Word, PowerPoint, or other help system can help reduce the Gulf of Evaluation, and the Gulf of Execution. For example, try to create multiple lines in an Excel file, or to create a string (like "1 2" or "p < .05") where the 1 and 2 and the "p" and "5" always stay on the same line.

12.6 Create six design dimensions related to anthropometric, behavioral, and social trade-offs. Provide an argument in each case as to why these are fundamental dimensions. The dimensions should cross the areas (i.e., note how they are related), or you should note why you think they are not relatable onto a single dimension

References

Besnard, D., Greathead, D., & Baxter, G. (2004). When mental models go wrong. Co-occurrences in dynamic, critical systems. *International Journal of Human-Computer Studies, 60*, 117–128.

Blackwell, A. (2002). First steps in programming: A rationale for attention investment models. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, 2–10. Arlington, VA: IEEE Press.

Blackwell, A., & Green, T. (2003). Notational systems—The cognitive dimensions of notations framework. In J. M. Carroll (Ed.), *HCI models, theories, and frameworks* (pp. 103–133). San Francisco: Morgan Kaufmann.

Blandford, A., Green, T. R. G., Furniss, D., & Makri, S. (2008). Evaluating system utility and conceptual fit using CASSM. *International Journal of Human-Computer Studies, 66*, 393–409.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2012). Discovering and analyzing usability dimensions of concern. *ACM Transactions on CHI, 19*(2), Article 9. 18 pages.

Green, T. R. G. (1989). Cognitive dimensions of notations. In *People and Computers V* (pp. 443–460). Cambridge, UK: Cambridge University Press.

Grudin, J. (1989). The case against user interface consistency. *Communications of the ACM, 32*(10), 1164–1173.

Jones, M. C., Churchill, E. F., & Nelson, L. (2010). Mashed layers and muddled models: Debugging mashup applications. In A. Cypher, M. Dontcheva, T. Lau & J. Nichols (Eds.), *No Code Required: Giving Users Tools to Transform the Web*: Burlington, MA: Morgan Kaufman.

Kotovsky, K., & Simon, H. A. (1990). What makes some problems really hard: Explorations in the problem space of difficulty. *Cognitive Psychology, 22*, 143–183.

Neisser, U. (1976). *Cognition and reality*. San Francisco, CA: W. H. Freeman.

Norman, D. A. (1988). *The psychology of everyday things*. NY: Basic Books.

Norman, D. A. (2009). *The design of future things*. NY: Basic Books

Norman, D. A. (2013). *The design of everyday things*. NY: Basic Books.

Peyton Jones, S., Blackwell, A., & Burnett, M. (2003). A user-centred approach to functions in Excel. In *ICFP'03: Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming* (pp. 165–176). ACM Press: New York, NY.

Rasmussen, J. (1983). Skills, rules, knowledge: Signals, signs and symbols and other distinctions in human performance models. *IEEE Transactions: Systems, Man, and Cybernetics, SMC-13*, 257–267.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*, 257–285.

Yeh, K.-C., Gregory, J. P., & Ritter, F. E. (2010). One laptop per child: Polishing up the XO laptop user experience. *Ergonomics in Design, 18*(3), 8–13.