First, it is often the case that design decisions made at the very beginning of the prototyping process are wrong and, in practice, design inertia can be so great as never to overcome an initial bad decision. So, whereas iterative design is, in theory, amenable to great changes through iterations, it can be the case that the initial prototype has bad features that will not be amended. We will examine this problem through a real example of a clock on a microwave oven.[2] The clock has a numeric display of four digits. Thus the display is capable of showing values in the range from `00:00` to `99:99`. The functional model of time for the actual clock is only 12 hours, so quite a few of the possible clock displays do not correspond to possible times (for example, `63:00`, `85:49`), even though some of them are legal four-digit time designations. That poses no problem, as long as both the designer and the ultimate users of the clock both share the knowledge of the discrepancy between possible clock displays and legal times. Such would not be the case for someone assuming a 24-hour time format, in which case the displays `00:30` and `13:45` would represent valid times in their model but not in the microwave's model. In this particular example, the subjects tested during the evaluation must have all shared the 12-hour time model, and the mismatch with the other users (with a 24-hour model) was only discovered after the product was being shipped. At this point, the only impact of iterative design was a change to the documentation alerting the reader to the 12-hour format, as it was too late to perform any hardware change.

The second problem is slightly more subtle, and serious. If, in the process of evaluation, a potential usability problem is diagnosed, it is important to understand the reason for the problem and not just detect the symptom. In the clock example, the designers could have noticed that some subjects with a 24-hour time model were having difficulty setting the time. Say they were trying to set the time for `14:45`, but they were not being allowed to do that. If the designers did not know the subject's goals, they might not detect the 24/12 hour discrepancy. They would instead notice that the users were having trouble setting the time and so they might change the buttons used to set the time instead of other possible changes, such as an analog time dial, or displaying AM or PM on the clock dial to make the 12-hour model more obvious, or to change to a 24-hour clock.

The moral for iterative design is that it should be used in conjunction with other, more principled approaches to interactive system design. These principled approaches are the subject of Part 3 of this book.

## 6.5 DESIGN RATIONALE

In designing any computer system, many decisions are made as the product goes from a set of vague customer requirements to a deliverable entity. Often it is difficult to recreate the reasons, or rationale, behind various design decisions. *Design*

2 This example has been provided by Harold Thimbleby.

*rationale* is the information that explains why a computer system is the way it is, including its structural or architectural description and its functional or behavioral description. In this sense, design rationale does not fit squarely into the software life cycle described in this chapter as just another phase or box. Rather, design rationale relates to an activity of both reflection (doing design rationale) and documentation (creating a design rationale) that occurs throughout the entire life cycle.

It is beneficial to have access to the design rationale for several reasons:

- In an explicit form, a design rationale provides a communication mechanism among the members of a design team so that during later stages of design and/or maintenance it is possible to understand what critical decisions were made, what alternatives were investigated (and, possibly, in what order) and the reason why one alternative was chosen over the others. This can help avoid incorrect assumptions later.
- Accumulated knowledge in the form of design rationales for a set of products can be reused to transfer what has worked in one situation to another situation which has similar needs. The design rationale can capture the context of a design decision in order that a different design team can determine if a similar rationale is appropriate for their product.
- The effort required to produce a design rationale forces the designer to deliberate more carefully about design decisions. The process of deliberation can be assisted by the design rationale technique by suggesting how arguments justifying or discarding a particular design option are formed.

In the area of HCI, design rationale has been particularly important, again for several reasons:

- There is usually no single best design alternative. More often, the designer is faced with a set of trade-offs between different alternatives. For example, a graphical interface may involve a set of actions that the user can invoke by use of the mouse and the designer must decide whether to present each action as a 'button' on the screen, which is always visible, or hide all of the actions in a menu which must be explicitly invoked before an action can be chosen. The former option maximizes the operation visibility (see Chapter 7) but the latter option takes up less screen space. It would be up to the designer to determine which criterion for evaluating the options was more important and then communicating that information in a design rationale.
- Even if an optimal solution did exist for a given design decision, the space of alternatives is so vast that it is unlikely a designer would discover it. In this case, it is important that the designer indicates all alternatives that have been investigated. Then later on it can be determined if she has not considered the best solution or had thought about it and discarded it for some reason. In project management, this kind of accountability for design is good.
- The usability of an interactive system is very dependent on the context of its use. The flashiest graphical interface is of no use if the end-user does not have access to a high-quality graphics display or a pointing device. Capturing the context in

which a design decision is made will help later when new products are designed. If the context remains the same, then the old rationale can be adopted without revision. If the context has changed somehow, the old rationale can be re-examined to see if any rejected alternatives are now more favorable or if any new alternatives are now possible.

Lee and Lai [209] explain that various proponents of design rationale have different interpretations of what it actually is. We will make use of their classification to describe various design rationale techniques in this section. The first set of techniques concentrates on providing a historical record of design decisions and is very much tailored for use during actual design discussions. These techniques are referred to as process-oriented design rationale because they are meant to be integrated in the actual design process itself. The next category is not so concerned with historical or process-oriented information but rather with the structure of the space of all design alternatives, which can be reconstructed by post hoc consideration of the design activity. The structure-oriented approach does not capture historical information. Instead, it captures the complete story of the moment, as an analysis of the design space which has been considered so far. The final category of design rationale concentrates on capturing the claims about the psychology of the user that are implied by an interactive system and the tasks that are performed on them.

There are some issues that distinguish the various techniques in terms of their usability within design itself. We can use these issues to sketch an informal rationale for design rationale. One issue is the degree to which the technique impinges on the design process. Does the use of a particular design rationale technique alter the decision process, or does it just passively serve to document it? Another issue is the cost of using the technique, both in terms of creating the design rationale and in terms of accessing it once created. A related issue is the amount of computational power the design rationale provides and the level to which this is supported by automated tools. A design rationale for a complex system can be very large and the exploration of the design space changes over time. The kind of information stored in a given design rationale will affect how that vast amount of information can be effectively managed and browsed.

### 6.5.1 Process-oriented design rationale

Much of the work on design rationale is based on Rittel's *issue-based information system*, or *IBIS*, a style for representing design and planning dialog developed in the 1970s [308]. In IBIS (pronounced 'ibbiss'), a hierarchical structure to a design rationale is created. A root *issue* is identified which represents the main problem or question that the argument is addressing. Various *positions* are put forth as potential resolutions for the root issue, and these are depicted as descendants in the IBIS hierarchy directly connected to the root issue. Each position is then supported or refuted by *arguments*, which modify the relationship between issue and position. The hierarchy grows as secondary issues are raised which modify the root issue in some way. Each of these secondary issues is in turn expanded by positions and arguments, further sub-issues, and so on.

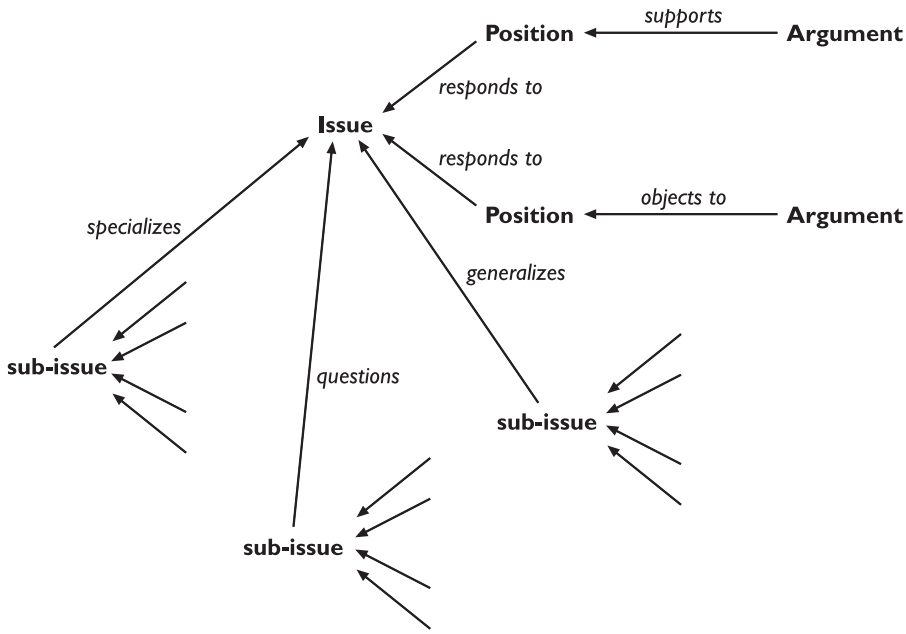


**Figure 6.8** The structure of a gIBIS design rationale

A graphical version of IBIS has been defined by Conklin and Yakemovic [77], called *gIBIS* (pronounced 'gibbiss'), which makes the structure of the design rationale more apparent visually in the form of a directed graph which can be directly edited by the creator of the design rationale. Figure 6.8 gives a representation of the gIBIS vocabulary. Issues, positions and arguments are nodes in the graph and the connections between them are labeled to clarify the relationship between adjacent nodes. So, for example, an issue can suggest further sub-issues, or a position can respond to an issue or an argument can support a position. The gIBIS structure can be supported by a hypertext tool to allow a designer to create and browse various parts of the design rationale.

There have been other versions of the IBIS notation, both graphical and textual, besides gIBIS. Most versions retain the distinction between issues, positions and arguments. Some add further nodes, such as Potts and Bruns's [297] addition of design artifacts which represent the intermediate products of a design that lead to the final product and are associated with the various alternatives discussed in the design rationale. Some add a richer vocabulary to modify the relationships between the node elements, such as McCall's Procedural Hierarchy of Issues (PHI) [231], which expands the variety of inter-issue relationships. Interesting work at the University of Colorado has attempted to link PHI argumentation to computer-aided design (CAD) tools to allow critique of design (in their example, the design of a kitchen) as it occurs. When the CAD violates some known design rule, the designer is warned and can then browse a PHI argument to see the rationale for the design rule.

The use of IBIS and any of its descendants is process oriented, as we described above. It is intended for use during design meetings as a means of recording and structuring the issues deliberated and the decisions made. It is also intended to preserve the order of deliberation and decision making for a particular product, placing less stress on the generalization of design knowledge for use between different products. This can be contrasted with the structure-oriented technique discussed next.

### 6.5.2 Design space analysis

MacLean and colleagues [222] have proposed a more deliberative approach to design rationale which emphasizes a post hoc structuring of the space of design alternatives that have been considered in a design project. Their approach, embodied in the Questions, Options and Criteria (QOC) notation, is characterized as *design space analysis* (see Figure 6.9).

The design space is initially structured by a set of questions representing the major issues of the design. Since design space analysis is structure oriented, it is not so important that the questions recorded are the actual questions asked during design meetings. Rather, these questions represent an agreed characterization of the

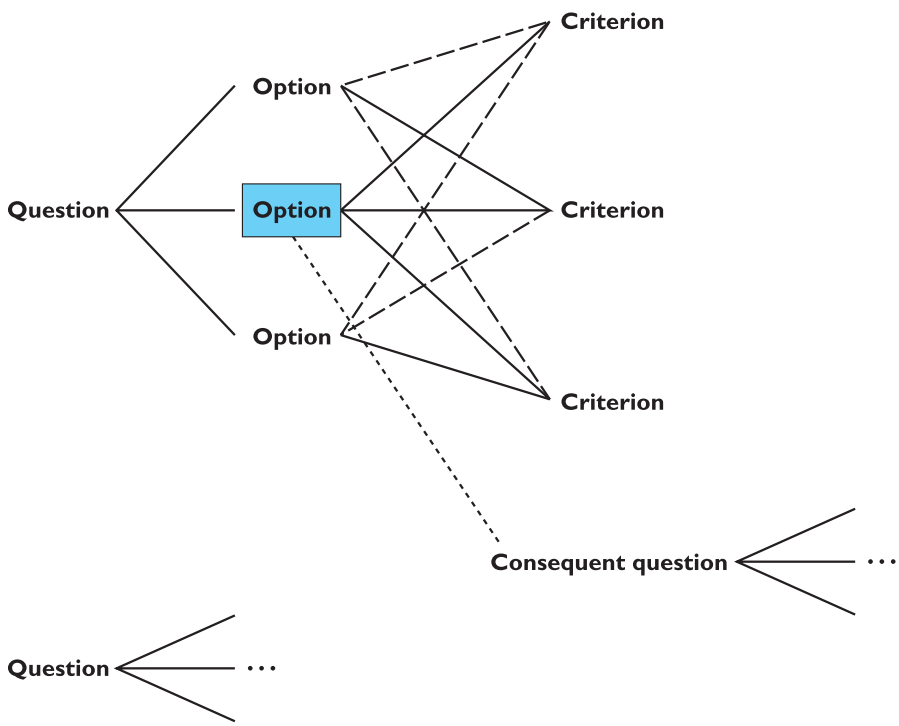


**Figure 6.9** The QOC notation

issues raised based on reflection and understanding of the actual design activities. Questions in a design space analysis are therefore similar to issues in IBIS except in the way they are captured. Options provide alternative solutions to the question. They are assessed according to some criteria in order to determine the most favorable option. In Figure 6.9 an option which is favorably assessed in terms of a criterion is linked with a solid line, whereas negative links have a dashed line. The most favorable option is boxed in the diagram.

The key to an effective design space analysis using the QOC notation is deciding the right questions to use to structure the space and the correct criteria to judge the options. The initial questions raised must be sufficiently general that they cover a large enough portion of the possible design space, but specific enough that a range of options can be clearly identified. It can be difficult to decide the right set of criteria with which to assess the options. The QOC technique advocates the use of general criteria, like the usability principles we shall discuss in Chapter 7, which are expressed more explicitly in a given analysis. In the example of the action buttons versus the menu of actions described earlier, we could contextualize the general principle of operation visibility as the criterion that all possible actions are displayed at all times. It can be very difficult to decide from a design space analysis which option is most favorable. The positive and negative links in the QOC notation do not provide all of the context for a trade-off decision. There is no provision for indicating, for example, that one criterion is more important than any of the others and the most favorable option must be positively linked.

Another structure-oriented technique, called Decision Representation Language (DRL), developed by Lee and Lai, structures the design space in a similar fashion to QOC, though its language is somewhat larger and it has a formal semantics. The questions, options and criteria in DRL are given the names: decision problem, alternatives and goals. QOC assessments are represented in DRL by a more complex language for relating goals to alternatives. The sparse language in QOC used to assess an option relative to a criterion (positive or negative assessment only) is probably insufficient, but there is a trade-off involved in adopting a more complex vocabulary which may prove too difficult to use in practice. The advantage of the formal semantics of DRL is that the design rationale can be used as a computational mechanism to help manage the large volume of information. For example, DRL can track the dependencies between different decision problems, so that subsequent changes to the design rationale for one decision problem can be automatically propagated to other dependent problems.

Design space analysis directly addresses the claim that no design activity can hope to uncover all design possibilities, so the best we can hope to achieve is to document the small part of the design space that has been investigated. An advantage of the post hoc technique is that it can abstract away from the particulars of a design meeting and therefore represent the design knowledge in such a way that it can be of use in the design of other products. The major disadvantage is the increased overhead such an analysis warrants. More time must be taken away from the design activity to do this separate documentation task. When time is scarce, these kinds of overhead costs are the first to be trimmed.

### 6.5.3 Psychological design rationale

The final category of design rationale tries to make explicit the psychological claims of usability inherent in any interactive system in order better to suit a product for the tasks users have. This psychological design rationale has been introduced by Carroll and Rosson [62], and before we describe the application of the technique it is important to understand some of its theoretical background.

People use computers to accomplish some tasks in their particular work domain, as we have seen before. When designing a new interactive system, the designers take into account the tasks that users currently perform and any new ones that they may want to perform. This task identification serves as part of the requirements for the new system, and can be done through empirical observation of how people perform their work currently and presented through informal language or a more formal task analysis language (see Chapter 15). When the new system is implemented, or becomes an *artifact*, further observation reveals that in addition to the required tasks it was built to support, it also supports users in tasks that the designer never intended. Once designers understand these new tasks, and the associated problems that arise between them and the previously known tasks, the new task definitions can serve as requirements for future artifacts.

Carroll refers to this real-life phenomenon as the *task–artifact cycle*. He provides a good example of this cycle through the evolution of the electronic spreadsheet. When the first electronic spreadsheet, *VisiCalc*, was marketed in the late 1970s, it was presented simply as an automated means of supporting tabular calculation, a task commonly used in the accounting world. Within little over a decade of its introduction, the application of spreadsheets had far outstripped its original intent within accounting. Spreadsheets were being used for all kinds of financial analysis, 'what-if' simulations, report formatting and even as a general programming language paradigm! As the set of tasks expands, new spreadsheet products have flooded the marketplace trying to satisfy the growing customer base. Another good example of the task–artifact cycle in action is with word processing, which was originally introduced to provide more automated support for tasks previously achieved with a typewriter and now provides users with the ability to carry out various authoring tasks that they never dreamed possible with a conventional typewriter. And today, the tasks for the spreadsheet and the word processor are intermingled in the same artifact.

The purpose of psychological design rationale is to support this natural task–artifact cycle of design activity. The main emphasis is not to capture the designer's intention in building the artifact. Rather, psychological design rationale aims to make explicit the consequences of a design for the user, given an understanding of what tasks he intends to perform. Previously, these psychological consequences were left implicit in the design, though designers would make informal claims about their systems (for example, that it is more 'natural' for the user, or easier to learn).

The first step in the psychological design rationale is to identify the tasks that the proposed system will address and to characterize those tasks by questions that the user tries to answer in accomplishing them. For instance, Carroll gives an example

of designing a system to help programmers learn the Smalltalk object-oriented programming language environment. The main task the system is to support is learning how Smalltalk works. In learning about the programming environment, the programmer will perform tasks that help her answer the questions:

- What can I do: that is, what are the possible operations or functions that this programming environment allows?
- How does it work: that is, what do the various functions do?
- How can I do this: that is, once I know a particular operation I want to perform, how do I go about programming it?

For each question, a set of *scenarios* of user–system behavior is suggested to support the user in addressing the question. For example, to address the question 'What can I do?', the designers can describe a scenario whereby the novice programmer is first confronted with the learning environment and sees that she can invoke some demo programs to investigate how Smalltalk programs work. The initial system can then be implemented to provide the functionality suggested by the scenarios (for example, some demos would be made accessible and obvious to the user/programmer from the very beginning). Once this system is running, observation of its use and some designer reflection is used to produce the actual psychological design rationale for that version of the system. This is where the psychological claims are made explicit. For example, there is an assumption that the programmer knows that what she can see on the screen relates to what she can do (if she sees the list of programs under a heading 'Demos', she can click on one program name to see the associated demo). The psychological claim of this demo system is that the user learns by doing, which is a good thing. However, there may also be negative aspects that are equally important to mention. The demo may not be very interactive, in which case the user clicks on it to initiate it and then just sits back and watches a graphic display, never really learning how the demo application is constructed in Smalltalk. These negative aspects can be used to modify later versions of the system to allow more interactive demos, which represent realistic, yet simple, applications, whose behavior and structure the programmer can appreciate.

By forcing the designer to document the psychological design rationale, it is hoped that she will become more aware of the natural evolution of user tasks and the artifact, taking advantage of how consequences of one design can be used to improve later designs.

**Worked exercise** *What is the distinction between a process-oriented and a structure-oriented design rationale technique? Would you classify psychological design rationale as process or structure oriented? Why?*

**Answer** The distinction between a process- and structure-oriented design rationale resides in what information the design rationale attempts to capture. Process-oriented design rationale is interested in recording an historically accurate description of a design team making some decision on a particular issue for the design. In this sense, process-oriented design rationale becomes an activity concurrent with the rest of the design

process. Structure-oriented design rationale is less interested in preserving the historical evolution of the design. Rather, it is more interested in providing the conclusions of the design activity, so it can be done in a post hoc and reflective manner after the fact.

The purpose of psychological design rationale is to support the task–artifact cycle. Here, the tasks that the users perform are changed by the systems on which they perform the tasks. A psychological design rationale proceeds by having the designers of the system record what they believe are the tasks that the system should support and then building the system to support the tasks. The designers suggest scenarios for the tasks which will be used to observe new users of the system. Observations of the users provide the information needed for the actual design rationale of that version of the system. The consequences of the design's assumptions about the important tasks are then gauged against the actual use in an attempt to justify the design or suggest improvements.

Psychological design rationale is mainly a process-oriented approach. The activity of a claims analysis is precisely about capturing what the designers assumed about the system at one point in time and how those assumptions compared with actual use. Therefore, the history of the psychological design rationale is important. The discipline involved in performing a psychological design rationale requires designers to perform the claims analysis during the actual design activity, and not as post hoc reconstruction.

## 6.6 SUMMARY

In this chapter, we have shown how software engineering and the design process relate to interactive system design. The software engineering life cycle aims to structure design in order to increase the reliability of the design process. For interactive system design, this would equate to a reliable and reproducible means of designing predictably usable systems. Because of the special needs of interactive systems, it is essential to augment the standard life cycle in order to address issues of HCI.

Usability engineering encourages incorporating explicit usability goals within the design process, providing a means by which the product's usability can be judged. Iterative design practices admit that principled design of interactive systems alone cannot maximize product usability, so the designer must be able to evaluate early prototypes and rapidly correct features of the prototype which detract from the product usability.

The design process is composed of a series of decisions, which pare down the vast set of potential systems to the one that is actually delivered to the customer. Design rationale, in its many forms, is aimed at allowing the designer to manage the information about the decision-making process, in terms of when and why design decisions were made and what consequences those decisions had for the user in accomplishing his work.

## EXERCISES

6.1 (a) How can design rationale benefit interface design and why might it be rejected by design teams?
(b) Explain QOC design rationale using an example to illustrate.

6.2 Imagine you have been asked to produce a prototype for the diary system discussed in the worked exercise in Section 7.2.3. What would be an appropriate prototyping approach to enable you to test the design using the usability metrics specified, and why?

## RECOMMENDED READING

J. A. McDermid, editor, *The Software Engineer's Reference Book*, Butterworth–Heinemann, 1992.
A very good general reference book for all topics in software engineering. In particular, we refer you to Chapter 15 on software life cycles and Chapter 40 on prototyping.

I. Sommerville, *Software Engineering*, 6th edition, Addison-Wesley, 2000.
This textbook is one of the few texts in software engineering that specifically treats issues of interface design.

X. Faulkner, *Usability Engineering*, Macmillan, 2000.
An excellent and accessible introduction to usability engineering covering, amongst other things, user requirements capture and usability metrics.

J. Whiteside, J. Bennett and K. Holtzblatt, Usability engineering: our experience and evolution. In M. Helander, editor, *Handbook for Human–Computer Interaction*, North-Holland, 1988.
The seminal work on usability engineering. More recent work on usability engineering has also been published by Jakob Nielsen [260, 261].

J. M. Carroll and T. P. Moran, editors, *Design Rationale: Concepts, Techniques and Use*, Lawrence Erlbaum, 1996.
Expanded from a double special journal issue, this provides comprehensive coverage of relevant work in the field.