Some Notes on Models and Modelling

Michael Jackson

Department of Computing, The Open University, UK
jacksonma@acm.org

Abstract. Analytical models are a fundamental tool in the development of computer-based systems of every kind: their essential purpose is to support human understanding and reasoning in development. To support reasoning, models must be substantially formal. The relationship between a formal model and its—typically—non-formal subject demands care: particular attention must be paid to the model interpretation, which maps its formal terms to the phenomena of the subject. An analytical model is to be regarded not as an assertion, but as a predicate within a larger logical structure of reasoning. Analogical models, such as databases, act as run-time surrogates for some parts of the problem world; in their design the properties of the model itself must be carefully distinguished from those of its subject. Some models may be informal: informal models have many legitimate uses, but cannot serve as a basis for formal reasoning.

1 Modelling and Understanding

The subject of these notes is the use of models and modelling in the development of computer-based systems. Models are of many kinds, built for many purposes. Whatever the explicit purpose of a model, a vital implicit purpose is always to achieve, record and communicate some human understanding of its subject—that is, of whatever is being modelled. The subject and its understanding provide the central theme of these brief notes.

2 Models and Subjects

Richard Feynman, the physicist, described how as a teenager he had thought about certain elementary problems in Euclidean geometry [1]. He manipulated the diagrams in his mind: he anchored some points and let others float, imagined some lines as stiff rods and others as stretchable bands, and let the shapes slide until he could see what the result must be. He was using a mental model of a physical system to help him to understand an abstract mathematical system. Teachers of elementary arithmetic do something similar when they use a real physical model of an abstract mathematical system—for example, helping children to understand integer multiplication and division by playing with rectangular arrays of pennies. In both cases understanding of a less familiar *subject* is achieved through the medium of a more familiar, and therefore more accessible, *model*.

The word ‘model’ has many meanings and shades of meaning, but in the meanings that are most interesting in the development of software and information systems it

A.T. Borgida et al. (Eds.): Mylopoulos Festschrift, LNCS 5600, pp. 68–81, 2009.

always denotes one role in a binary relationship: the complementary role is 'subject'. The essence of this relationship is that in some way the model captures or expresses some understanding or knowledge of the subject, and can be used to provide further information or insight about it. As Feynman's pivoting rods and the arithmetic teacher's arrays of pennies show, there is no *a priori* requirement that the model be more abstract than the subject. It can be more concrete; but it must be in some way simpler, more accessible, more familiar, or more tractable than the subject. If it is not, the model achieves little or nothing: it can be discarded and the subject explored directly.

3 Mental Models and Reasoning

The model's superior accessibility, familiarity and tractability depend, of course, on the knowledge and experience of the people who construct and use it. An adult with a modest knowledge of arithmetic can dispense with the physical arrays of pennies, although they may still occasionally be useful in imagination to furnish a mental model. A number theorist will expect to think more abstractly, and perhaps more formally. Whether the number theorist is doing something radically different from forming and using a mental model appears to be an open question in cognitive science.

In *Mental models: a gentle guide for outsiders* [2] P N Johnson-Laird writes: "Our plan in the rest of this paper is to start with how models can be used to reason, and to contrast them with the orthodox view that reasoning is based on a sort of mental logic. ... On one side, there are those ... who claim that it depends on formal rules of inference akin to those of a logical calculus. On the other side, there are those, such as ourselves ..., who claim that it is a semantic process that depends on mental models akin to the models that logicians invoke in formulating the semantics of their calculi." It is, of course, perfectly plausible that human reasoning depends both on formal inference and on the use of mental models, combined according to the problem to be solved and the innate and learned capabilities, knowledge, and inclinations of the reasoner.

Johnson-Laird reports a reliable experimental difference between the following two questions. People find it harder to answer one of them correctly than the other:

> "If A then B; A is true; what can be said about B?"

(to which the correct answer is "B is true") and

> "If A then B; B is false; what can be said about A?"

(to which the correct answer is "A is false"). The second is the contrapositive of the first, and for someone with a little knowledge of elementary logic is of precisely equivalent difficulty. Yet the first question is answered correctly by nearly everyone, while a substantial minority of people fail on the second question, and those who do succeed take reliably longer to answer. Those who claim that reasoning depends on rules of inference identify a longer chain of deductions for the second question. Those who claim that it depends on mental models cite a *principle of truth*:

> "Individuals tend to minimise the load on working memory by constructing mental models that represent what is true, but not what is false."

Evidently, human capacity for understanding is not itself easy to understand.

4 Overt Models

A *mental model* is a private possession, held in its owner's mind and revealed only at the owner's choice—and then only indirectly and uncertainly. An *overt model*, by contrast, is a public possession, intended to capture, communicate and make more widely available some understanding or notion of its subject, or some information about it.

Russell Ackoff distinguishes [3] three kinds of model used in natural science: iconic, analogue and symbolic. An *iconic* model looks like its subject. A photograph is an iconic model of its human subject, and a child's toy car is an iconic model of its subject automobile. An *analogue* model represents its subject by exhibiting different, but analogous, properties. The classic example is the use of a hydraulic system—water flowing in a complex of pipes—as an analogue model of an electrical circuit. The volume of water flowing, the narrowness of the pipes, and the water pressure are analogues of the electrical current, resistance, and potential difference respectively. Iconic models are always, to at least some small extent, also analogue models. A *symbolic* model, as its name implies, represents its subject symbolically, the symbols occurring as elements in some formalism that allows direct formal reasoning and calculation. Thus a system of equations may be a symbolic model. By solving the equations for given values of some variables the values of others can be determined. A symbolic model may be representable as an analogue model: for example, an equation may be represented by a graph.

In software development, which is a particular kind of engineering rather than a natural science, a slightly different taxonomy of models is useful. First, because engineering is concerned with specific artifacts and specific cases, we must distinguish *specific* from *generic* models. A commonly familiar example of the distinction is that between a class model and an instance model in object-oriented development. The class model is generic: its subject is those properties that are shared by all instances of the class. The instance model is specific: its subject is one instance only.

Second, the particular power of computers to create and maintain internal data structures make it both necessary and illuminating to distinguish *analytical* from *analogical* models. An analytical model is a *description* of its subject; in Ackoff's taxonomy it may be iconic or symbolic. For example, a finite state machine model of a possible behaviour of interest may be represented iconically in a diagram or symbolically in a transition table. Analytical models are most often used in the development process itself to help the developers to capture, understand and analyse properties of the problem world, the system requirements, and the software. A curve-fitting program, that chooses a curve type and adjusts the parameters of the curve to fit a set of data points, can be regarded as creating an analytical model at system execution time, but in most application areas this is exceptional. Because the properties of interest in software development are most often those that hold—or should hold—for all executions of the system, analytical models are typically generic. A finite state machine model, for example, may describe all successful withdrawals of cash from all possible ATMs in a particular system. Where some part of the problem world is a singleton and has only static properties—for example, the road layout for a particularly complex junction at which traffic is to be controlled—an appropriate analytical model may be specific rather than generic; but this is somewhat unusual.

An analogical model in software development is always specific. It is not used in the development process, but in the system operation: the analogical model is built and maintained by execution of the hardware/software machine. An analogical model is not a description of its subject, but a concrete thing in its own right. It is a data structure represented on a substratum of computer storage—most often as a collection of disk records or an assemblage of programming objects in RAM. The commonest example of such a specific analogical model is a database held on disk. The system continually maintains the database, using information directly input during system operation. The database can then act as an easily accessible *surrogate* for its subject, allowing the system to provide needed or requested information by examining the state of the database.

5 Model Imperfection

An overt analytical model is a description of its subject. The description may be formal or informal. It may be expressed in text, in equations, in diagrams, or in any other way judged suitable to the content of the description. An analytical model is inherently non-physical. It can be represented in a physical medium—for example, written on paper or encoded in a computer file; but the physical medium is not the model, just as the plastic disc of a CD is not the music. In a broad sense, this intangibility makes the model invulnerable to the vicissitudes of time and physical failure. The model can describe change over time, but it is not itself subject to change; it can describe physical decay and failure, but it is not itself subject to decay or failure.

However, this immunity to decay does not guarantee the quality of an analytical model *qua* model. For a non-formal subject, such as an engineering artifact, any analytical model—certainly at the levels of granularity of interest to software developers—is at best an approximation to its subject's actual physical behaviour and properties.

An analogical model, like an analytical model, is necessarily an imperfect approximation to its subject; but it is also imperfect in an additional way. Not only is the underlying understanding of the subject's properties inevitably an approximation, but the analogical model itself possesses phenomena and properties which have no counterpart in the subject and may distort understanding of the analogy. In the classic analogue model, we observe that a broken pipe leaks water: so we might—quite wrongly—infer that a broken wire will leak electricity into the air. In the same way, an analogical model incorporated into a computer-based system possesses phenomena and properties which have no counterpart in the subject. For example, a relational database may have null values in some row-column intersections; the rows may be ordered and indexed; and rows may be deleted for reasons of managing the database resources. These phenomena and properties can distort understanding of the analogy that underpins the relationship between model and subject.

6 Models and Interpretations

The meaning of the information provided by a model about its subject depends on an *interpretation*: that is, on an agreed mapping between the elements of the model and

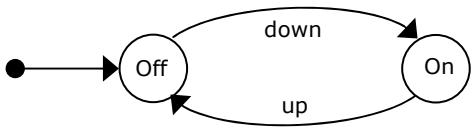

Fig. 1. An analytical model

the elements of the subject that they denote. Figure 1 shows an analytical model in the form of a state machine diagram having labelled circles for states and labelled arrows for transitions between states:

The arrow from the small solid circle points to the initial state. The interpretation must map the model's state labels to identifiable states of the physical subject, or its transition labels to identifiable events, or both. There are therefore at least these three components involved in using an analytical model: the subject matter; the model; and the interpretation.

The subject of this model may be the controlling switch of an electrical device. The interpretation may then be:

Off ≈ *state*: the switch is off
On ≈ *state*: the switch is on
down ≈ *event*: the switch is flipped down
up ≈ *event*: the switch is flipped up

The apparent simplicity of the model and interpretation may conceal some potential uncertainties. For example:

- The *On* and *Off* states may be independently observable phenomena of the subject. A more informative interpretation might then have been:

  Off ≈ *state*: no current can flow through the switch
  On ≈ *state*: current can flow through the switch

  Alternatively, it may be that the *On* and *Off* states are not independently observable phenomena of the subject, but are defined by the model:

  Off $\stackrel{\text{def}}{=}$ *state*: either no *up* or *down* event has yet occurred, or else the most recent such event was an *up* event
  On $\stackrel{\text{def}}{=}$ *state*: some *up* or *down* event has occurred, and the most recent such event was an *up* event

- The model may describe a switch, like an old-fashioned tumbler switch, in which two successive *up* events cannot occur without an intervening *down* event, and vice versa. Nothing is said about the effect of an *up* event in the *Off* state or a *down* event in the *On* state, because they cannot occur.

  Alternatively, the switch may be spring-loaded to return to a central position on each flip, placing no constraint on the sequence of *down* and *up* events. Nothing is

said about the effect of an *up* event in the *Off* state or a *down* event in the *On* state, because they do not alter the switch state.

- Identifying a formal term with a physical event or state is in itself an abstraction. For example, treating *down* and *up* as atomic events abstracts from the complex physical processes they involve. In a tumbler switch, moving the knob compresses a spring as the knob moves towards the centre of its travel; when the knob passes the centre and moves to the end of its travel, the compressed spring exerts gradually increasing force on the switch contacts, eventually pulling them open and swinging them to their other position.

  Abstracting this process as an atomic event is a good choice if the switch operates fast enough and reliably enough for the purposes of the model being built.

- *Off*—whether defined or independently observable—is identified as the *initial* state, but the term *initial* has been given no interpretation. It may, for example mean:

  Initial ≈ the state in which the newly manufactured switch leaves the factory

  or:

  Initial ≈ the state of the switch when the system begins execution

In a particular use of the model, the meaning of *Initial* may be given by the context in which the model is proposed. Whether their meanings are given by the model context, in an interpretation, or—as too often—left implicit, failure to deal properly with initial states is a rich source of error in software development. The problem of uninitialised variables is well-known in programming; but it is more difficult, and more often neglected, in the development of computer-based systems. The essence of the difficulty is to ensure compatibility between the initial state of the software to be executed and the current state of the problem world.

7 Designations

An element of an interpretation that associates a formal term in the model with a class of observable phenomena of the subject has been called [4] a *designation*. A designation must give a clear enough rule for recognising the phenomena to avoid harmful uncertainty. What is harmful depends on the nature and bounds of the subject, on the purpose to which the model will be put in the development, and on the opportunities that the developed system will offer for human common sense to override potential system failures resulting from modelling errors. In traditional manual systems, based on written processes and rules, system defects can often be repaired by reasonably resorting to available exception procedures when the system would otherwise deliver absurd results. To the extent that a computer-based system aims at automation it excludes such exception procedures. It is therefore important that developers' analytical models should correspond very closely to the subjects they describe.

A good correspondence between model and subject requires care in choosing and distinguishing the different subject phenomena to be designated. Consider, for example, a designation of the term *mother*:

```
mother(m,p) ≈ m and p are human beings and m
                is the mother of p
```

For a system concerned with Old Testament genealogy this designation is adequate: within the scope of the human beings mentioned in the Old Testament the meaning of "*m* is the mother of *p*" is perfectly clear. For a system to manage the administration of a kindergarten it may perhaps be good enough, provided that no special treatment is needed for adoptive mothers and stepmothers. In a fertility research clinic this designation would be useless: it would be necessary to distinguish natural mothers, genetic mothers, surrogate mothers and, perhaps, others.

If previously existing systems in the application area have a low degree of automation, the prevailing terminology of the area is not necessarily an adequate guide: application experts may underestimate the extent to which exceptional procedures are regularly invoked to compensate for terminological uncertainty. One well-known illustration is the phenomenon of a telephone *call*. Suppose that A phones B, but B is busy. A accepts the system's offer to connect them later, when B is no longer busy, and A hangs up. Soon, A's phone rings, and A picks it up and hears the ringback tone. After a short period of ringing, B answers and A and B talk. Is this one, two, or three *calls*? Reliance on the obsolete notion of a 'call' caused much difficulty in computer-based telephone systems in the last quarter of the twentieth century [5]. Similarly vague terminology is found in many application areas where the prevailing terminology includes obsolete relics of an earlier, much simpler, system.

8 Interpretation for Analogical Models

Interpretation for an analogical model is significantly more complex than for an analytical model. At first sight it may seem that the same notion of interpretation will serve for analogical as for analytical models: an interpretation maps the terms of the model to the phenomena of the subject. However, there is an important difference. An overt analytical model is itself a description, expressed in some chosen language, using a finite number of terms. An interpretation maps just those terms to the elements of the subject. Physical phenomena of any tangible representation of the model are to be ignored. For the state machine shown in Figure 1 we do not seek an interpretation of the lengths of the arrows or the diameter of the circles: the semantics of a description in the chosen graphical language are unaffected by those graphical phenomena.

An analogical model, by contrast, is a physical—and therefore inevitably non-formal—thing: it is a concrete structure of phenomena, not an abstract structure of formal terms. It does not embody any clear distinction between those of its own phenomena that are intended to participate in the analogy and those that are not. In effect, to understand the analogical model we also need an explicit analytical model.

Figure 2 shows how model, subject, and interpretation are related for a simple analytical model and for an analogical model.

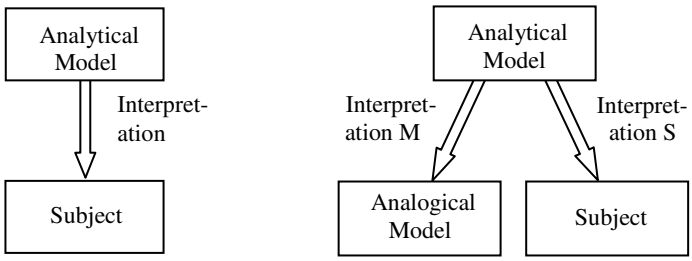

Fig. 2. Interpreting Analytical and Analogical Models

The purpose of the analytical model in the right side of Figure 2 is to bound the physical properties of interest in the analogical model, by pointing out the analogies that relate it to the subject. This analytical model therefore has two distinct interpretations: *S* interprets it as a model of the subject; the other, *M*, as a model of the analogical model. The analogy is in this sense indirect: an exact understanding rests on the analytical model that the analogical model shares with its own subject.

9 Designing and Understanding an Analogical Model

In developing any model—whether analytical or analogical—the conceptual starting point must be to consider what questions the model is intended to answer, and what properties it must therefore capture. In both cases, these are properties of the subject of the model, not of the model itself.

For an analytical model, the subject properties are directly expressed in the model. For an analogical model, the development process is conceptually more complex. First, the subject must be understood, and an appropriate analytical model developed. Then a type of analogical model—perhaps a database, or an assemblage of objects, or more generally a data structure in the programming language—is chosen or designed to offer exact analogues of the properties of the analytical model. Of course, there is a difficulty here. Because the analogical model is a concrete thing in its own right, and will inevitably possess properties that have no analogue at all in the subject, or impose constraints that are not present in the subject, the analogy is inevitably imperfect. So a part of the design task is to find good a compromise between fidelity to the subject and efficiency in the representations and accesses that the model affords. This task demands some clarity of thought: in particular, it demands a clear distinction between the properties of the model and the properties of the subject.

There is an obvious temptation to save time and effort by abbreviating the development task, short-circuiting the two interpretations and developing the analogical model directly from the subject. This is a common approach in object-oriented modelling, in which the developer describes the subject domain as if it were itself an assemblage of objects communicating by sending messages to each other. The benefit of the approach is a certain economy and directness. The disadvantage is an increased risk of error: the analytical model disappears from view, and is considered only tacitly and sporadically during the design of the analogical model.

10 The Context of a Model

A model has a purpose, in the sense that it is intended to capture certain properties that its subject may have, and to answer certain questions about them. But it also has a *context*, in a broader sense. For an analogical model in a computer-based system the context is, essentially, always the same. The computer continually collects information about the subject in the problem world, perhaps analyses and summarises it in some way, and updates the model to reflect it. The analogical model can then serve as a surrogate for the world: the state of the model is an acceptably accurate analogue of the state of the world. Its context is implicit: the presence of the model constitutes an assertion that the analogous properties hold in the subject.

For an analytical model, by contrast, there are many possible contexts. An analytical model can be thought of a predicate *M* applied to its subject *S*: *M(S)* holds if—and only if—the model is a true description of the subject. Just as a predicate can appear in a sentence, so too can an analytical model. For example, in a system to control a lift, the problem world *W* consists of the building's floors and lift shafts, the electromechanical lift equipment, the users, the request buttons and display lights, and so on. We can imagine three distinct analytical models of the problem world:

- *G(W)*: this model captures the *given* properties of the problem world. It describes, for example, the arrangement of the floors, the causal chain between the motor setting and the behaviour of the lift car, the way the lift and lobby doors work, and so on.
- *R(W)*: this model captures the *required* properties of the problem world, that the computer must somehow enforce. It describes, for example, the property that if the *Up* button at a floor is pressed, the lift car will eventually arrive at the floor and the doors will open, and that if a button inside the lift is then pressed that corresponds to a higher floor the lift car will go to that higher floor, and so on.
- *C(W)*: this model captures the behaviour at the *computer's* interface with the problem world. It describes, for example, the property that if the *Up* button at floor 3 is pressed when the lift is stationary at the ground floor, then the motor direction is set to *Up* and the motor is switched *On* (by the computer), that when subsequently the sensor at floor 3 is set *On* (by the arrival of the lift car) the motor is then switched *Off* (by the computer), and so on.

In a successful development, the relationship among these three models is something like:

$$(\mathrm{G(W)} \wedge \mathrm{C(W)}) \Rightarrow \mathrm{R(W)}$$

That is: the given properties of the problem world, in conjunction with the additional properties due to its interaction with the computer, ensure that the requirement is satisfied. The *truth* of each model depends on the changing context. At the outset of the development, *G(W)* is true (assuming that the building and the lift equipment are known and correctly described). At that point in time, however, *R(W)* nor *C(W)* is true. *C(W)* is not true because the computer has not yet been built and installed; and *R(W)* is also not true—in the absence of the computer it is merely what the customer wishes were true. Later, when the development has been successfully completed, and the software is executing as intended, all three models will be true.

11 The Local Context of a Model

The formula $(G(W \wedge C(W)) \Rightarrow R(W)$, interpreted for a whole system, expresses a global context for the three models: it encompasses the whole development, and the behaviour of the whole system. In the course of the development it will be necessary, for a realistic system, to decompose all three models in a way that is not necessarily simple. These decompositions aim to master the complexity of the development problem and the eventually developed system, reducing one large and unmanageably complex problem to a structure of simpler problems.

The decompositions that will be most useful will depend on the problem in hand. Because realistic systems are complex in a heterogeneous way, the most useful decompositions will be similarly heterogeneous. Some candidate dimensions of decomposition are:

- Decomposition by function or feature. An e-commerce system for consumer use has such features as shopping basket, credit card validation and charging, collaborative filtering, shipping management, and so on.
- Decomposition by operational phase. An avionics system must behave differently in the different phases of a flight: in taxiing, taking off, climbing, cruising and so on.
- Decomposition by problem world conditions. The behaviour of an air traffic control system in normal conditions is different from its behaviour in an emergency. A lift control system providing normal lift service must behave differently when an equipment fault—such as a failing hoist motor—is detected.

Decomposition can be usefully regarded as decomposition into subproblems, in which each subproblem defines a local context. Different subproblems have different requirements; they need different software behaviours for their solution; they concern different parts or *domains* of the problem world; and they exploit different properties of those domains. To understand and analyse a subproblem it is necessary to practise a separation of concerns. The models needed for a subproblem are local models for local contexts.

The local context of a model puts in place a set of local assumptions, determining its purpose and the content of the description it embodies of its subject. In a lift system, for example, the local context of providing normal lift service assumes that the equipment functions normally, exhibiting the behaviour that is necessary if the lift car is to be sent from floor to floor, the doors opened and closed appropriately, and so on. The relevant model of the lift equipment is therefore a model of *healthy* lift equipment, describing the normal functioning. By contrast, the local context of fault monitoring requires a model of *dubious* lift equipment, describing equipment that may or may not function normally, and focusing on the properties that allow faults to be detected and perhaps diagnosed, when they occur.

12 The Scope and Span of a Model

The *scope* of a model is the set of all phenomena denoted by its interpreted terms. The scope of the model of the control switch shown in Figure 1 is {*up, down, On, Off*} if

they are all designated, independently observable, phenomena; but if *On* and *Off* are defined in terms of *up* and *down*, then the scope is only {*up, down*}. The switch may have other phenomena—for example, it may have a rotary dimmer knob—but they are *out of scope*: the developer of the model has decided that for the purpose of the model they should be ignored.

The *span* of a model is how much it describes, measured by time or space or any other relevant quantifiable dimension. For example, in a lift control system it may be appropriate to model the required opening and closing of the lift doors in normal operation. One required property may be the opening and closing of the doors when the lift car serves a floor, including the behaviour when an obstruction is detected: a model of this property has a span of one visit to one floor by one lift. Another required property may be that the lift doors are never opened when the car is not positioned at a floor: a model of this property has a span of one lift over the whole local context of normal operation. A model whose span is one lift can, of course, be applied to all the lifts in the system; and a model whose span is one visit to one floor can be applied to all visits to all floors; the spans of the models themselves, however, are not affected by their larger application.

To be intelligible, the scope and span of a model must be appropriate to its subject and content. A notable—and widely practised—way of obfuscating a model is to replace it by a loose collection of models of inappropriately small span. For example, a state machine may be fragmented into a distributed collection of individual transitions. Each transition makes understandable sense only as a part of the whole state machine: taken alone it can be understood only by a reader who already possesses a firm and clear mental model of the whole machine, and has this mental model vividly in mind while reading each fragment. The obvious danger is that neither the writer nor the readers have such a mental model available, and the fragments are never brought together to validate their collective meaning. The result is a model prone to many errors. For example: omitting transitions that should be included; making false assumptions about reachability; and ignoring the disastrous effect of a neglected, but possible, sequence of transitions.

The appropriate span for a model is not always obvious. Whenever the behaviour of a system feature or function is arranged in sporadic or cyclic episodes—for example, in use cases—it is naturally attractive to construct a model of the function with a span of one episode. For some aspects of the episode this will be entirely appropriate. Much of the interaction of a bank customer with an ATM is encapsulated within the episode, and should be modelled with that span: the card is inserted before the PIN is entered; the card is withdrawn before the money is delivered; and so on. However, the episode may include events of a more global import: a certain amount of money is withdrawn from the account; a new PIN is specified; a new cheque book is requested. These events belong to behaviours of spans larger than the episode. Depending on the complexity of these behaviours, it may be necessary to model them also, each in its appropriate span.

Using an appropriate span for a model is not just a matter of bringing together enough information in one document. A good model answers its designed questions in the simplest possible way, laying the smallest possible burden on the reader's powers of perception, memory and reasoning, and helping the reader to form a good mental model of the subject. Because short-term human memory is severely limited, this

process of assembling a mental model must not require information to be collated from many separate places. From a purely formal point of view, a graph may be equally well represented in a diagram or in a list of nodes and arcs; but from the point of view of human intelligibility the diagram is hugely superior for almost every purpose. The most important questions to be answered by a graph model are not usually questions about isolated nodes or arcs: they are questions about traversals of paths in the graph. The primary significance of a node or arc is its role and position in a set of possible traversals. The core success of the famous London Underground map is that it makes traversals—train journeys—very easy to identify. The importance of traversals explains also why even those computer scientists who disdain diagrams prefer to write their programs as structured, indented texts.

13 Vague Models

Almost any overt description—diagrammatic, textual or numerical—can be regarded as a generic or specific analytical model of its subject. Whether it is a useful model for its intended purpose will depend on many considerations. The importance of explicit designation for ensuring that the meanings of the terms used in the model are clear and exact has been stressed in this essay; but a shopping list can be useful even if some its entries are vague: "Lots of oranges if they're sweet", or even "Something nice for our dinner". A more structured checklist can be useful even if it is similarly vague: "The chief quality measures are high customer satisfaction and a low rate of operator error". The shopper and the project manager know that the terms in their models are very imprecise, and are careful not to lay too much weight on them. It is sometimes—but not always—worthwhile to establish quantitative empirical criteria for these imprecise terms.

A model can be vague for other reasons than the absence of designations of its terms. The descriptive language itself may use fuzzy general notions like "about" and "some" and "low rate"; some linguistic terms—such as "nice" and "satisfaction" may have no clear meaning that can be designated; some operators or connectives in the language—such as the lines or arrows in a graph, or the various node symbols—may have no clear semantics. This kind of vagueness can be easily tolerated, and can even be helpful, in contexts in which the model plays the role of a personal reminder, a sketch for live discussion, or an informal private note between two people; but it is very damaging if the model's purpose is to serve as a basis for any kind of reasoning. Any conclusions reached by reasoning about a model must be encashed by interpreting them in terms of the phenomena and relationships of the subject: if this cannot be done reliably then the value of the conclusions is diminished accordingly.

14 Building Precision on Vagueness

Formal reasoning cannot be based on an informal model. A faulty map cannot be corrected until two sources of faults have been eliminated. First, the cartographic conventions must be clearly established—for example, whether a road bridge over a railway is distinguished from a rail bridge over a road, and, if so, how. Second, the

designations must be clarified—for example, does a cross signify only a church or can it also signify a mosque or synagogue? Then the map can be corrected by comparing it with the terrain it is intended to describe and modifying the map to correspond to the terrain. Similarly, the informality of a model in software development cannot be repaired without repairing the inadequacies both of the modelling language and of the designations that relate the model to its subject.

Suppose, for example, that a model is concerned with relationships of *dependency* among a population of distinct specific things or tasks or goals or documents. Such a model may be useful in program design, in tracing the relationship of an implementation to its requirements, and in other contexts too. In program design a relevant designation might be:

depends(m,n) ≈ *m* and *n* are program modules and *m* depends on *n* in the sense that *m* will not function correctly unless *n* functions correctly

This designation may seem clear enough. The writer or reader of the model may even be tempted to infer that depends is transitive:

$\forall$ m,n,o • depends(m,n) $\wedge$ depends(n,o) $\Rightarrow$ depends(m,o)

However, perhaps the designation is far from clear enough. Suppose that modules in this context are procedures, that interaction is procedure call, and that a module *m* has functioned correctly if the result of '*m(p1,p2,...)*', including any side effects, satisfies the specification of *m*. Then to clarify the meaning of *depends(m,n)* it may be necessary to consider these and other possibilities:

- When *m* is called, it may, or may not, call *n* before returning to its caller.
- When *m* is called, it always calls *n* before returning to its caller.
- When *m* is called, it always calls *n* before returning to its caller, but *m* does not use any result of the call (for example, *n* simply logs the call).
- For some calls of *n*, *n* fails to satisfy its specification, but none of the calls for which it fails can be a call by *m*.
- *n* calls *m*, and *m* can satisfy its specification only if *n* executes an appropriate sequence of calls with appropriate arguments (for example, *m* is an output module encapsulating a file and requiring the sequence of calls

  <m('open'); m('write',v)*; m('close')>).

- *m* and *n* both call a third module *q*, and *q* can satisfy its specification only if it is called by an appropriate sequence of calls with appropriate arguments, calls by *m* and calls by *n* being interleaved.

It may be possible—or impossible—to provide a clear designation of *depends(m,n)* in the particular subject to be modelled. If it is impossible, there is no point is building an edifice of formal reasoning on such shaky foundations. It is not required that every useful model be formal and exact; but the writer and reader of a vague or informal model should avoid the mistake of treating it as a basis for formal reasoning.

15 Summary

These notes have briefly discussed several aspects of models and modelling in the context of software development. Their unifying theme is a pair of relationships in which any model must participate. One relates the overt model to the human understanding—that is, to the mental model—that it seeks to express or evoke. The other relates the overt model to its subject matter in the physical and human world. In effect, these two relationships unite to form a bridge between the world and our understanding of it. An effective practice of modelling must seek to create a bridge that is strong at both ends: it must find abstractions of reality that are adequate for the purpose in hand and its context; and it must express and convey those abstractions in ways that serve the goal of human understanding as well as possible.

Because computer programs are, in effect, formal and exact processes, they admit no vagueness in their execution. We must therefore be doubly confident in the formal models of the world on which we base our software development. As John von Neumann pointed out [6]:

"There is no point in using exact methods where there is no clarity in the concepts and issues to which they are to be applied."

References

1. Gleick, J.: Genius: Richard Feynman and Modern Physics. Little Brown (1999)
2. Johnson-Laird, P.N., Girotto, V., Legrenzi, P.: Mental models: a gentle guide for outsiders. Sistemi Intelligenti (1998)
3. Ackoff, R.L.: Scientific Method: Optimizing Applied Research Decisions. Wiley, Chichester (1962)
4. Jackson, M.: Software Requirements & Specifications: A Lexicon of Practice, Principles, and Prejudices. Addison-Wesley, Reading (1995)
5. Zave, P.: 'Calls Considered Harmful' and Other Observations: A Tutorial on Telephony. In: Margaria, T., Steffen, B., Rückert, R., Posegga, J. (eds.) AIN 1997, ICALP-WS 1997, VISUAL-WS 1998, ACoS 1998, and ETAPS-WS 1998. LNCS, vol. 1385, pp. 8–27. Springer, Heidelberg (1998)
6. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press, Princeton (1944)