9

Design Education

The demands upon students of engineering design are intense. They need to understand analysis of engineering systems, synthesis of functional solutions, and to be familiar with many computational tools. They are challenged now to put product generation and systems design into the contexts of international, environmental, economic and political spheres wherein the design has to operate. All that has to be carried out in a climate of increasing competition and reduced life cycle time. The design process is rapidly becoming computer-based rather than computer-aided, by which is meant design must be carried out at a level that cannot be done other than with computer modelling. The growing complexity of products and the trend towards concurrent engineering in design all reinforce that trend. What would be of great help in an increasingly computer literate student body is for the very process of using computer systems to encourage a proper understanding of the design process.

In this chapter we explore ways that definitive methods may be used to help students and others towards computer-based modelling and towards self-teaching by interaction, animation and prototyping.

9.1 The Educational Context of Design

9.11 Historical Background

Historically, innovation has been thought of as a kind of amateur game. In both Britain and America, myths about inventiveness such as "necessity is the mother of invention" and "Watt and his steaming kettle" tended to reinforce that idea, despite the fact that innovation was fed by solid scientific discovery and technological change. The separation of innovation from science led inevitably to the separation of industry from academia.

As science grew so it became the province of specialists separating into departments of physics, chemistry, geology and so on. The result was that the brilliant achievements of gifted mechanics and engineers such as Maudsley, Nasmyth and Whitworth were based upon basic training and apprenticeships.

When by the middle of the nineteenth century Britain noticed that other countries were more successful technically, the answer was sought by introducing Engineering into the Universities. The effect was not as great as expected! After the 1st World War the government took over much of the industrial research because of industry's reluctance to innovate and the growing gap between science and industry. The pattern of modern education was set. The authors in [Burns and Stalker, 1986] put it like this:

> "Two major changes have occurred in the social circumstances affecting the production of innovations. First, industrial concerns have increased in size: greater administrative complexity has brought in a wide range of bureaucratic positions and careers. Their positions make it imperative that innovation was seen to come from within not by newcomers.
>
> The other change has occurred in the form of institutional relationships within which innovation had been possible. The familiar and social circumstances typical of the eighteenth century provided the ease of communication necessary for the major synthesis of ideas and requirements that introduced the early revolutionary inventions. In the nineteenth century new institutional forms introduced barriers between science and industry. By the twentieth century the new and elaborate organizations of professional scientists has been matched by one of technical innovators into groups overlapping teaching and research institutions, Government departments and industry"

The institutionalisation of design has not proved to be helpful because of that separation of product and process. It led directly to the idea of Engineering Science, the ultimate separation! Design was interpreted as a branch of analysis. Indeed the author's own experience is of a "Design" course in which a clutch is "designed" from its description. What was asked for was the calculation of the clutch plate size, an analytical problem with the design taken out.

9.12 Learning the Design Process

In the last twenty years Design teaching has undergone a renaissance in both government and academia, following various reports such as those by Bullock on Academic Enterprise and by Feilden on Engineering Education. Enormous effort has gone into trying to understand the design process and to find better ways of

teaching it to students. However "standard" approaches to teaching design still appear to collude with the idea that there is a sequence of activities that will lead inevitably toward a "correct" solution. A typical student text has the suggestion that the first three stages of the design process are called “Problem Finding”

> “1. IDENTIFY the fundamental need to be satisfied
> 2. DEFINE the precise problem arising from that need
> 3. PARAMETERS: state the constraints within which any solution must fit” [Starkey, 1992]

The desire to structure design has carried over into computer aided design. Students need to know that many computer tools over-constrain the designer. Those kinds of tools are designed to solve particular problems and the inputs must be precisely defined. Indeed there are those who see design as needing such constraints. Some of those making computer aids for manufacturing would like to constrain the designer by limiting the features that can be part of a design, for example to those that can be made with current technology. The danger in limiting innovation is clear. That kind of bottom-up approach may be helpful in many detail design problems but one should recognise that one of the reasons for a design aid being made at all may have been that it could be made with the computational tools available, not that it was necessarily the most important. (As often happens in life, we try to solve the problems that look tractable and ignore the intractable ones and hope they go away!)

The design process is much more elusive than implied by these "steps to a solution". It is interesting that practising designers do not identify with any of the so-called design process descriptions beloved of academics. A recent (mischievous!) comment by Allan Gardam, Chief Mechanical Engineer at Pilkington Optronics, was that the best description of the design process is represented by a single block diagram.

That comment is supported by work done by [Kelly, *et al.*, 1986] who comment

> "As we reviewed the various theories and models, we began to realise that in almost every major innovation of recent times each functional phase is linked in some way to the others: every phase in our block diagram has lines connecting it to and from every other block in the diagram. Instead of a linear-sequential picture .. we had a plate of spaghetti and meatballs!"

All is not lost however. We can identify some important ingredients of design, particularly at the conceptual stage. The most important of these has already been discussed at length: namely observation and experiment, for which the EdenLisp is designed. Trial and error are the very stuff of design, and of science itself. The act of finding out has still that charm, often indeed thought of as mere playing. Sir Hermon Bondi makes this observation in the Foreword to [Michie, 1986].

> " I myself was involved in space affairs when in April 1970, a serious malfunction in the Apollo 13 mission to the Moon led to great anxiety for the safe return of the crew. By a rapidly devised brilliant strategy, the crew returned to earth safe and sound, albeit without landing on the Moon. When I expressed my astonishment at the speed the solution had been found, I was told that the staff at Mission Control had been spending their time playing games with the equipment and that rescue from disaster was one of the games! Our play instinct is always something to be fostered."

Play is of course not totally unstructured. As one finds something that amuses or interests it is investigated more thoroughly, an approach that has its counterpart in design. It is that which researchers into learning have found to be most significant in gaining and retaining knowledge. Taking one extreme, the effort required to retain small amounts of "nonsense syllables" was found to be excessive because there was no relation to prior knowledge. In real-life, as we have found in the discussion on Minsky, knowledge is "chunked" into percepts that relate common observations. The question is then how new knowledge gets chunked. [Wærn, 1989], in discussing general learning principles, shows that the most successful learning situations are top-down; they arise from linking new knowledge with prior knowledge and then being able by reflection to discriminate and then to generalise. Discrimination consists for example of a child seeing cows and horses and not calling them both "bears", the only prior concept she had for large animals. It is the process of seeing what is different and what is similar in the new situation. Generalisation is the chunking stage, associating similarities.

Both discrimination and generalisation refer to *declarative* material. The results of learning declarative material are always expressed declaratively. We have to fetch the material directly from memory and reproduce them. *Procedural* knowledge on the other hand has to do with associating knowledge. It is a much more difficult learning operation with three stages. First there is the cognitive stage: the declarative knowledge, then an associative stage, putting knowledge together in sequence, and finally the autonomous stage where the knowledge becomes chunked. The first stage is easiest and learning is fastest. Learning then drops off rapidly.

The lesson from this discussion is clear. A top-down, declarative approach is most fruitful both for learning *about* design and for learning *to* design. We can therefore benefit from some of the research into the design process namely the notion of hierarchical decomposition.

9.2 Learning to Design

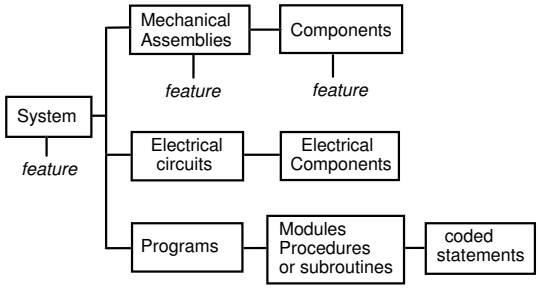
9.21 Hierarchical Decomposition

Decomposition is the first stage in the top-down approach, as has already been discussed in chapter 2. Ullman suggests how it helps to structure one' s thinking in arriving at the requirement.

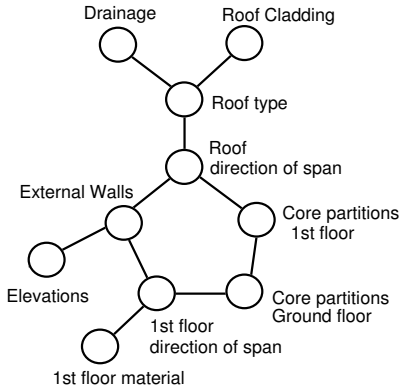
> "In general, during the design process, the function of the system and its decomposition is considered first. After the function has been decomposed into the finest subsystems possible, assemblies and components are developed to provide these functions. Thus a hierarchy of mechanical is shown in the top row of *[the figure reproduced as fig 9.1a]*. Also shown in this figure is one further decomposition of mechanical objects" [Ullman, 1992, *chapter 2*]

Sometimes the problem is not that easy to structure hierarchically in Ullman' s way. For example, compare Ullman' s nice hierarchy*fig 9.1a* with the cyclic problem in *fig 9.1b* that [Cross, 1989] raises, where the problem is of a particular house design detail identified by [Luckman, 1984].

> "Architects identified five decision areas concerned with the directions of span of the roof and first floor joists, and the provision of load bearing or non-load-bearing walls and partitions. Making a decision in one area had implications in other areas that had implications in further areas, in one case coming full circle."



a) Decomposition of Design Disciplines from [Ullman, 1992]



b) Decomposition Structure of House Design, from [Cross, 1989]

Figure 9.1 Decomposition Problems

However even here, Cross argues, cycles can be avoided by strategic choices and despite such difficulties most design problems can usefully be decomposed in a hierarchical manner.

We have shown in the examples in chapter 7 that the definitive method is both declarative in form and admirably suited to hierarchical decomposition. Initial statements of a design problem can be written in EdenLisp as definitions in quite vague terms such as the following.

```
CarEngine = f(EngineType, TransmissionType, maxPower, maxWeight)
EngineType = choice_of(diesel, petrol_injection, petrol_carb)
MaxPower = powerRange(max, min)
```

Each of the definitions becomes the starting points for the specification, initially without any defined variables. Functions would need to be defined but may be quite simple selection functions such as `choice_of`. (In EdenLisp there would have to be type declarations, but that too becomes a useful conceptual exercise, thinking about what the parameters would be in a design and sorting out the important from the less important or downstream variables.) As the specification gains detail certain parameters will acquire values or a range of values that define the "design space" delineating possible designs.

What is instructive is that the specification of the design in definitive terms requires the designer to decompose the tasks and suggest related tasks and possible solution spaces. As those definitions take the form of lists, they are open ended and invite addition and redefinition - an interaction that is vital at the initial formulation of the design problem. Second, that decomposition helps to identify the possible agents interacting in the design process. Third, the designer also begins to identify abstractions in the design. In order to arrive at sufficiently general definitions of particular relationships it is necessary to think quite hard about the patterns that underlie the design spaces. That is apparent in the dental plate design described in some detail in chapter 7. Patterns are identified that make possible alternative designs easy to generate; sets of relations become separate scripts quite naturally and so enable agents to be teased out. We therefore conclude that EdenLisp provides a structuring method for design that is natural, interactive and generic.

9.22 Design Folio

Having ascertained the specification and the main hierarchies of sub-problems the designer proceeds, according to [Starkey, 1992], to the next phase called "Problem solving".

"4. Create ideas for alternative solutions
5. Evaluate each of the created ideas
6. Isolate the preferred solution
7. Implement that solution."

Again the phraseology seems deceptive. The casual reader might think that by putting heading 4 under Problem Solving the author is intending to show that given sufficient preparation in steps 1 to 3 the designer can converge on a "solution". It sometimes happens that no solution is possible to the problem as put. It would not be a useful exercise, for example, to design a lathe that can machine a high precision spindle of 0.3 mm diameter to 0.001 mm, whilst on the same machine be capable of machining a shaft of 600 mm diameter. In such cases a revision of the basic requirement is called for. It is necessary for the student to regard a cyclic or iterative approach to be the norm at any of the stages, rather than design being seen as a sequential stream of processes.

A further ambiguity in the idea of design being problem solving is that it may be perceived in terms of analytical tractability. In all but the most trivial of designs, because of the infinite variety of choices, the known information is small compared with what is unknown: rather like having 100 simultaneous equations and values for only 20 of the 100 variables. Any analytical model of the design is therefore going to be limited. We explored this point in conceptual terms in chapter 2. It is essential that the student cultivates an approach that bears these difficulties in mind. It is in that frame that we suggest that the Definitive method, perhaps in the form of EdenLisp can help.

We showed in chapter 8 that we can create with Definitive methods a computational object as a Virtual Prototype. Using the decomposition model described above we can develop the design by analogy with the non-computational approach. The designer tries out a number of different ideas, developing some of them to a degree that shows their feasibility. Those ideas will go into the design folio. In a similar way, the student could develop a number of definitive scripts as candidates for the Virtual Prototype, these being stored in an equivalent "design folio", probably in the form of a library of files in a directory. That computational design folio is not simply a set of library files on the same topic, as for example the collection of generics in PADL-2 or the storage of partial solutions in the

SDRC I-DEAS CADCAM software. The key difference is that in those systems the composition of partial solutions has to be done by the user providing the connections interactively. In EdenLisp the connections can be made by means of guarded actions. Although the user' s reasoning process may be the same in each case, it is made explicit on the Virtual Prototype. And because relationships are based upon real-world observations, the reactions of the VP to any further alterations are more related to the way a physical object would behave.

From the educational point of view the discipline of needing to tease out the inter-relationships between the different ideas is exactly what one would like to see made explicit. The student has to decide upon the relationships to be made and in order to do that is forced to "think aloud" about the design problems. Furthermore the decisions needed at the milestones are those the designer needs to make in progressing the design in non-computational methods. The application example of the dental plate design shows how that pattern-making process operates in practice. The use of EdenLisp in education would therefore encourage the student to adopt a realistic approach to decision making and encourage greater insight into the design possibilities

9.3 Parametric Studies

9.31 Sensitivity Study

Analysis of designs has long been the domain of the computational engineer, that phase being the easiest to automate. The value of EdenLisp is that it enables experiments to be carried out that may or may not be analytically tractable. Designers are always having to deal with partial solutions where it is necessary to "suck it and see". That process is easy with EdenLisp as it will ignore partial solutions that cannot be solved and present to the user those relationships that do have values. Because of that users can play around with values of parameters in design relationships in order to get a "feel" for the way that those parameters behave. Different values of variables in the domain of the designer are quickly entered and the effect observed. Manipulating EdenLisp scripts can enable sensitivity analysis to be more flexible than conventional optimisation techniques: observations may be made on the effect of incremental changes in parameters and also the effect on the constraints imposed on the optimisation.

By way of example we investigate the optimisation procedures for the selection of sizes for a compression spring suggested by [Siddall, 1982]. Loading is static compression and the spring ends are assumed to be closed and ground flat.

Notation

N = Number of active coils (usually end coils are inactive, so N = no. coils - 2)
D = Mean diameter of the coil (mm)
d = Diameter of the wire used for the spring (mm); usually a preferred standard size.
G = Bulk Modulus (MPa)
S = Maximum shear stress of spring material (MPa)
F_{max} = Maximum working Load (N)
l_{max} = Maximum free length (mm)
D_{max} = Maximum coil diameter (mm)
d_{min} = Minimum wire diameter (mm)
F_p = Preload compression force (N)
C_f = End Coefficient of the spring (= 1 for parallel ends with one fixed, one free)
δ_{pm} = Maximum allowable deflection under preload (mm)
δ_w = Deflection from preload position to maximum load position (mm)

1. Criterion Function

The optimisation criterion is that of minimising the volume of the spring wire used.

$$U = \frac{\pi^2}{4} Dd^2 (N + 2)$$

This criterion is subject to a series of 8 constraints Φ_i as follows.

2. Constraints

1. Strength. The shear stress in the spring must be less than the yield shear strength of the spring material, S_{shear}. The stress has two components: a shear force on the cross-section of the wire, and torsion of the wire. The total shear stress in the wire is expressed in terms of the loading and a spring index C_f, a function of *D/d.* The stress constraint Φ_1 is

$$\Phi_1 = S_{shear} - 8C_f F_{\max} \frac{D}{\pi d^3} \geq 0$$

2. Deflection constraint

The stiffness of a coil spring K (N/mm) is

$$K = \frac{Gd^4}{8ND^3}$$

The deflection (mm) of the spring under maximum static load is $\delta = F_{max}/K$. The spring length under load F_{max} is 105% of the solid length the spring. The free length is given by

$$l_f = \delta + 1.05(N + 2)$$

The deflection constraint is

$$\Phi_2 = l_{\max} - l_f \geq 0$$

3. Wire Diameter constraint

The wire diameter must not be less than the minimum specified.

$$\Phi_3 = d - d_{\min} \geq 0$$

4. Coil Diameter

The outside diameter of the coil must not exceed the maximum

$$\Phi_4 = D_{\max} - D - d \geq 0$$

5. Coil Winding restriction

The mean coil diameter must be at least three times the wire diameter to prevent it being too tightly wound.

$$\Phi_5 = C - 3 \geq 0$$

6. Preload deflection

The preload deflection must be less than F_p/K and so the constraint is

$$\Phi_6 = \delta_{pm} - \delta_p \geq 0$$

7. Total deflection constraint

The combined deflection must be consistent with the free length of the unloaded spring

$$\Phi_7 = l_{\max} - \delta_p - \frac{(F_{\max} - F_p)}{K} - 1.05(N + 2) \geq 0$$

8. Specified deflection

The deflection from the preload position to the maximum load position must be

$$\Phi_8 = \frac{(F_{\max} - F_p)}{K} - \delta_w \geq 0$$

We do a parameter study by entering the equations as EdenLisp definitions, and the constraints as `Phi1`, `Phi2`, *etc.* as definitions with violation messages. Drawings of the spring under the conditions applied are easy to do by associating

the variables with appropriate geometrical models. Possible EdenLisp script and some results follow.

```
;;; EDENLISP Exercise in Optimising compression spring dimensions
;;;
;  Definitions of relationships
Vol   = pi^2*D*d^2*(N+2)/4
K     = G*d^4/(8*N*D^3)
Lf    = dw + 1.05*(N+2)
delpm = Fp/K

; definitions of constraints
Phi1  = S - 8*Cf*Fmax*D/(pi*l^3)
Phi2  = Lmax - Lf
Phi3  = d - dmin
Phi4  = Dmax - D - d
Phi5  = C - 3
Phi6  = delpm - delp
Phi7  = lmax - delp - (Fmax-Fp)/K - 1.05*d*(N+2)
Phi8  = (Fmax-Fp)/K  - dpm

constraint1 = if Phi1 >= 0 then print "Phi1 -ve, Stress too high"
constraint2 = if Phi2 >= 0 then print "Phi2 -ve, Spring solid with no load"
constraint3 = if Phi3 >= 0 then print "Phi3 -ve, spring wire dia too small"
constraint4 = if Phi4 >= 0 then print "Phi4 -ve, Coil dia exceeds design max"
constraint5 = if Phi5 >= 0 then print "Phi5 -ve, Coil dia too small"
constraint6 = if Phi6 >= 0 then print "Phi6 -ve, Preload too great: spring
solid"
constraint7 = if Phi7 >= 0 then print "Phi7 -ve, Combined deflection too great"
constraint8 = if Phi8 >= 0 then print "Phi8 -ve, Load too great: spring solid"
```

```
;;; Results of two sets of values of variables d, D, N and material
```

Strength: S	1100	Stiffness: K	96.02
Elastic Modulus: E	205000	Deflection: δ	46.24
Bulk Modulus: G	80000	Free Length: Lf	259.39
Force: Fmax	4440	Delp	13.87
Length:Lmax	355	CF	1.58
Wire diameter: Dmin	5		
Outer diameter: Dmax	75	Optimisation: Volume	73629.59
Preload: Fp	1332	Constraints to be ≥ 0	
Preload def: dpm	150	Phi1	6.28
Deflection: dw	32	Phi2	95.61
End Coefficient: CE	1	Phi3	2
var: d	7	Phi4	47
var: D	21	Phi5	0
var: N	27	Phi6	136.13
constant: pi	3.1416	Phi7	0
Ratio D/d: C	3	Phi8	0.37

Strength: S	676.69	Stiffness: K	97.35
Elastic Modulus: E	207000	Deflection: δ	45.61
Bulk Modulus: G	80000	Free Length: Lf	354.62
Force: Fmax	4440	Del: p	13.66
Length:Lmax	355	CF	1.55
Wire diameter: Dmin	5		
Outer diameter: Dmax	75	Optimisation: Volume	182991.0
Preload: Fp	1330	Constraints to be ≥ 0	

```
Preload def: dpm            150      Phi1                    2.31
Deflection: dw               35      Phi2                    0.38
End Coefficient: CE           1      Phi3                       4
var: d                        9      Phi4                      38
var: D                       28      Phi5                    0.11
var: N                     30.7      Phi6                  136.34
constant: pi             3.1416      Phi7                       0
Ratio  D/d: C              3.11      Phi8                   -3.06

"Phi8 -ve, Load too great: spring solid"
```

Optimisation may be carried out as a batch process using numerical methods such as Siddall suggests in his text (*op cit.*). The advantage of interactive study of sensitivity is that the student can observe what happens to the optimisation function as different parameters are manually varied. Changing the values of variables in EdenLisp just involves redefinition, so the designer can quickly study the behaviour of important parameters and can just as easily change constraints. The tables show the results of varying the material strength, wire diameter and coil outer diameter. By checking what is happening to the constraints at the same time it is possible to see whether the constraints themselves are reasonable. When the value `Phi8 = - 3.06` is obtained in the second table, indicating a constraint violation, we can not only to vary the main parameter to get us out of violation but also see whether the specified deflection from pre-load position to maximum load position δ_w needs to be changed. It is difficult for a preconceived optimisation analysis to anticipate those kinds of adjustments to constraints; and for the student to know what to do with a result of an optimisation analysis when constraints are not examined in that way.

9.32 Parametric design

A second way that EdenLisp helps in parametric studies is in discovering relationships that represent combinations of parameters in a design. In our search for methods of analysing the shaft described in chapter 4 we showed that the relationships between segments of the shaft could be expressed in a hierarchical way by series of matrices that were peculiar to the geometry and material properties of each segment. Whilst identifying the components of the hierarchy, it became clear that certain combinations of parameters were important, as for example *L/EI* (symbols are respectively segment length, Young' s Modulus and Second moment of area of section). What we can then do is analyse those combinations across different designs and different materials, maximising or minimising the parameter combination according to the design criteria. These

parameter combinations are probably best expressed dimensionlessly, and many studies have been undertaken to help students understand and use for example particular material property combinations. Sensitivity studies of these parametric combinations is equally easy to do with EdenLisp

9.4 Self-Teaching

9.41 Animation

One method of animation in CAD is to take snapshots of the screen as slides in sequence and then show them quickly to create the illusion of animation. Alternatively a simple shape may be displayed on the fly, deleted and re-displayed in the new position. That second method is intrinsic to EdenLisp inasmuch as the automatic recomputation that takes place on redefinition not only overwrites previous parameter values with the new values, it also deletes any graphic entities that are changed as a result, and reconstructs the object in its new geometry.

An example of a self-learning script (written in DoNaLD) was discussed in principle in chapter 4, namely Bow' s Notation for bending of beams. In the diagram of *fig 4.7* reproduced and modified below as *fig 9.2* the three components of the notation are shown: beam loading, the vector diagram, and the polar diagram. These are connected by a graphical construction as follows.

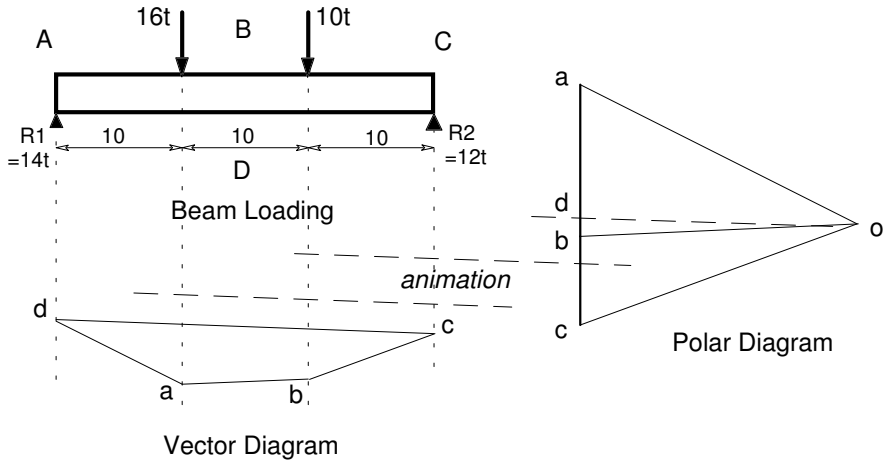


Fig. 9.2 Bow's Notation Animation

Bow' s notation:
On the Beam Loading diagram, label spaces between the forces. So, `A` is in the space between the 16t load and the reaction `R1`; `D` is between the two reactions and so on.

Construct the polar diagram from vectors that sum the known loads. So, vector ab represents the 16t load between A and B, vector bc the 10t load between B and C. In the diagram the loads are vertical but they need not be. Select any point not on the line as the pole o and join oa, ob and oc.

Translate vectors oa, ob and oc to fill the spaces labelled A, B and C projected down from the beam loading diagram in the area designated as vector diagram. Join c to d representing the resultant of the vectors. Translate vector cd back to the polar diagram to pass though pole o. Line od cuts the vector abc at point d. ad and dc represent reaction vectors R1 and R2, measured from the diagram as 14t and 12t.

In seeking to teach Bow' s Notation it is useful to animate the construction of each stage of the diagram. In the DoNaLD code for example, once the line dc is constructed in the vector diagram in the manner described, the code causes an animation of the translation of dc to the polar diagram. Displaying intermediate positions of the translation is done by redefining the vector dc for selected points along its path. The system re-evaluates the vector each time, drawing it in the new position after deleting the previous one. Unlike the picture in *fig 9.2* the vector is only seen instantly in one position but it appears to "move" across from the vector diagram to the polar diagram. Similarly the vector ad and dc detach from the polar diagram and translate to the reactions R1 and R2 and their magnitudes get written as labels.

As explained in chapter 4 DoNaLD is rather a clumsy tool for that animation and EdenLisp is more versatile because of the way it uses AutoLisp functions. To show that difference, a construction was made of a small bench vice in three dimensions. Animation of the movement of the handle may be achieved by a series of redefintions of its angular position, causing the vice grips to appear to move. The application to mechanisms and loci are obvious, and as observed in chapter 4 the student has full control over all the variables, including the presentation of the display itself.

9.42 Authoring

EdenLisp can make use of AutoCAD' s own system to help the student learn to use AutoCAD itself (a rather nice case of self reference!). The method is extensible to any situation where annotated graphics and text are required. We thus have a possible authoring system.

The method is to use AutoLisp functions accessed by EdenLisp that use the AutoCAD system of Programmable Dialogue Boxes, described in their Customisation Manual [AutoDesk, 1992]. The Dialogue Control Language (DCL) provided by AutoDesk allows one to program pop-up boxes, typically to display text or graphics and to provide buttons, sliders, highlighters, lists, toggles. Whatever is designed to go into pop-up boxes may be under the control of EdenLisp. Text that helps the user to see what is going on can be popped onto the graphics screen at the position where it does most good, for example to point to some aspect of the display that needs explanation.

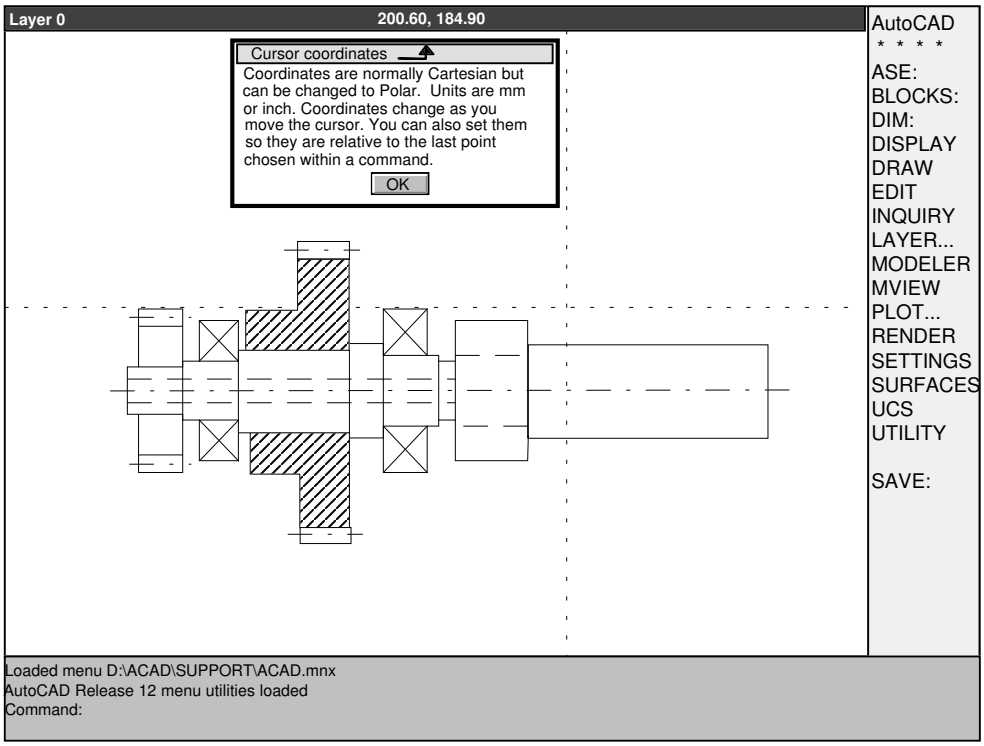


Fig 9.3 Example of using Dialogue Boxes from EdenLisp

Fig 9.3 illustrates the point. The dialogue box is created by putting the text of the box in one definition and then calling the DCL function to define the position and shape of the dialogue box. In the diagram the box has a single OK button that has to be clicked with the mouse to proceed. Other buttons, sliders, *etc.* can be similarly programmed and the results of touching those buttons can be monitored resulting in changes in the current state of the scripts. As the user changes the screen by selecting different commands it is possible that new help files can be popped up. Currently the method is driven as an embryo teaching aid, showing that it can be done in principle. Further development it necessary to make it a usable package