BUTTERWORTH HEINEMANN

Computer Integrated Manufacturing Systems Vol. 8, No. 2, pp. 105–115, 1995

0951-5240/95 $10.00 + 0.00

0951-5240(95)00004-6

The helical approach to software design

A W S Ainger* with the support of F Schmid†

**Human Centred Systems Ltd, Beaumont, Burfield Road, Old Windsor, Berks SL4 2JP, UK*
†*Advanced Railway Research Centre, University of Sheffield, Regent Court, Sheffield S1 4DA, UK*

This paper is used to outline a relatively recent approach to the development of software products. Practical experience of employing the traditional waterfall lifecycle model, the Spiral Model and concurrent engineering approaches in both small and large (pan-European) software projects provide the foundation on which to present, discuss and propose a new lifecycle model; the Helical LifeCycle Approach. The authors of the paper distinguish between prototypes, and model and postulate the need for the formalization of a new software engineering job role which is focused around the Helical Project LifeCycle.

Keywords: software design, Helical Project LifeCycle

Background

In 1979 a report (*Figure 1*)[1] concluded that only 2% of software supplied was usable as delivered, and a massive 47% of software was delivered but never used. These figures emanated from the US Department of Defense some 14 years ago, so it could be assumed that in the intervening years, with the advent of computer-aided everything (CAx), things would have changed. The situation has indeed changed, from the 2% success rate in 1979 to, in 1991, a 99% failure rate! (*Figure 2*). It is realized that the comparisons are not exactly like with like; however, a general trend can be identified, that of the generally poor performance, as far as the user is concerned, of software systems.

This general failure to meet user needs first time round accords with our own experience. Although the success rate of our projects (within a multi-million pound international organization) appeared to be significantly higher than those experienced in the US, the overall performance was still felt to be low. As a result, an extensive internal survey of over 100 projects over a 10 year period was undertaken. In the survey we attempted to identify critical success factors within software projects. Many parameters were monitored, such as: hardware platform; software languages used; size of the software team; qualification and experience of the development team; overall cost of the project; geographical position of the final installation; size of the project, etc. No significant correlation between any of these factors and the more successful projects were identified.

During further analysis it was found, almost by accident, that there was a small correlation between the number of meetings held and successful projects. This initially, appeared as a surprise, as it was commonly understood that the fewer meetings there were the better the project progress. However, on more detailed analysis it was determined that it was not the number of meetings that gave the highest correlation with successful results, but the *type* of meetings, and who was present at those meetings. Regardless of hardware or

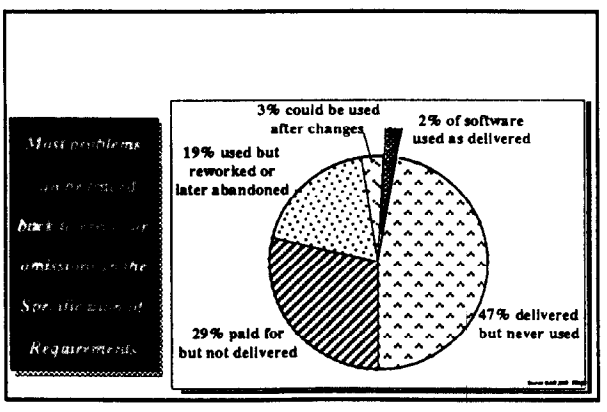

Figure 1 Only 2% of software supplied was usable as delivered (1979)

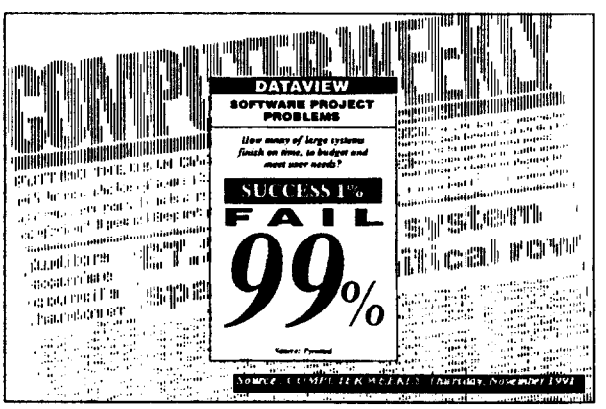

Figure 2 Failure rate of 99% (1991)

software platforms, regardless of the qualifications of the development team, and regardless of the cost of the project, it appeared that the most successful projects had significantly more meetings (both formal and informal) with the users of the system than those that did not. There was, however, another correlation that was even higher than the user meetings. It was found that failed projects followed more rigorously than others the waterfall project lifecycle (*Figure 3*).

At that time (mid-1980s) it was something of a revelation to find evidence that pointed towards the demise of the waterfall project lifecycle. More recently, however, there have been a number of papers[2–4] that, to quote Butler[5], indicate that 'too much order can mean chaos'. Even so, the failure of the traditional waterfall project lifecycle has not yet been widely recognized. There would still appear to be many prestigious companies and organizations that adhere to the waterfall project lifecycle[6].

It was in the early 1980s that work commenced on what is now termed the Helical Lifecycle Approach. This is best explained by using the principles of concurrent engineering. The initial work was conducted under a European Strategic Research and Development into Information Technology (ESPRIT) Project[7]. The Project, entitled 'Human-Centred CIM Systems', developed further Dr Cooley's statement that 'Human Centred CIM (Computer Integrated Manufacture) is a new approach to CIM, where the system is designed around human beings and integrates human capabilities, skills inventiveness, etc.' The £5.6 million project involved six organizations from three European countries, and is believed to be the first pan-European project to research human centred CIM systems.

Concurrent engineering

The benefits of concurrent engineering would appear to be obvious (*Figure 4*). Just by overlapping the traditional project development stages, (that is, requirements specification, design, build and implementation), significant time benefits would appear to accrued. However, it is only when the detail is examined that the impracticalities of this approach emerge (*Figure 5*). For example, if the project team is half-way through specifying a systems product, it would appear that the design work, if started before the specification work is complete, could be wasted, as one could 'start to design a car before discovering that what you really need is a bicycle'[8].

A more practical approach to concurrent engineering could be structured as in *Figure 6*, where an initial *global* look at the specification is made prior to any initial design work. Then, only after the final specification work is completed, can the design work start again, but in far more detail. It has been likened to 'first looking at a map of the terrain and then getting out the mountain bike'[8]. As we tend to view life sequentially, particularly so in the field of systems design, a sequential view of concurrent engineering can be made

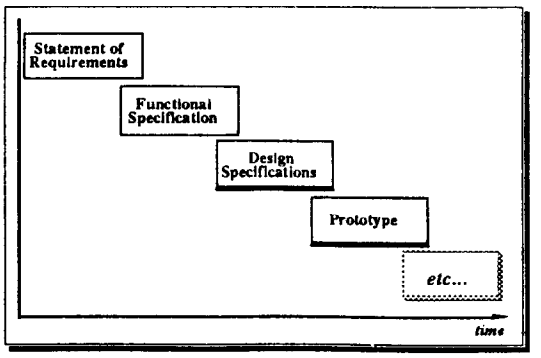

Figure 3 Traditional waterfall project lifecycle

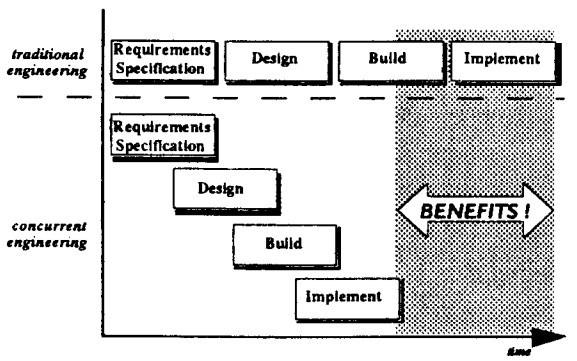

Figure 4 The benefits of concurrent engineering

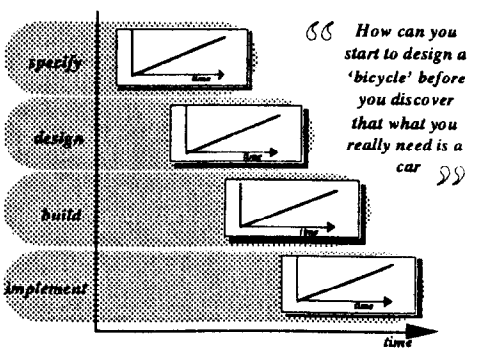

Figure 5 The impracticalities of concurrent engineering!

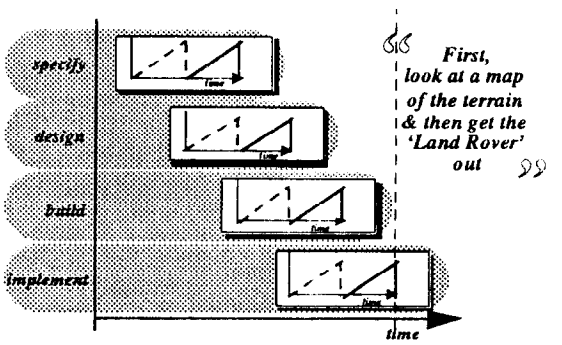

Figure 6 A practical approach to concurrent engineering?

(*Figure 7*). If we look at timeslices through the overlapping traditional waterfall lifecycle method, we find we have the apparently impossible task of: first partially specifying the system; then finally specifying the system whilst simultaneously partially designing it; followed by finally designing it and partially simultaneously building it; followed by finally building it and partially implementing it, etc!

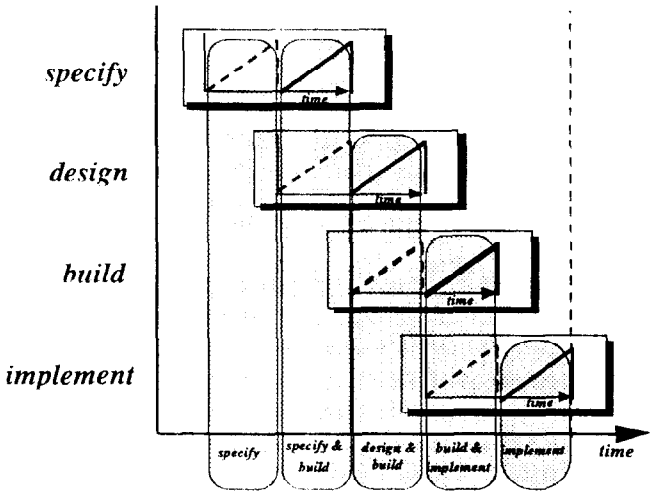

Figure 7 A 'sequential' view of concurrent engineering

In theory, this 'saw tooth' approach can be taken to the limit (*Figure 8*). Here the overall time savings can be enormous. There is, however, a problem. How does one obtain, in practice, the benefits of this sequential view of the concurrent engineering aspects when applying the traditional waterfall lifecycle approach? An answer can be seen in *Figure 9*, a representation based on a design helix.

In postulating an underlying sequential pattern in the concurrent engineering approach, we have found a practical solutions to the software design problem based on the production of a number of *models*. In the traditional waterfall cycle, this equates to the visualization of the prototype as a virtually complete entity. If we make models, and many of them, it is possible to achieve high levels of feedback early on the design process, thereby ensuring the convergence of views between the users and the systems designers. It is the method by which these multiple models are produced that has been termed the 'helical approach' (*Figure 10*). Such visualizations of the potential functionality of the product at the different stages of the development cycle are often described as 'cardboard models' in the sense

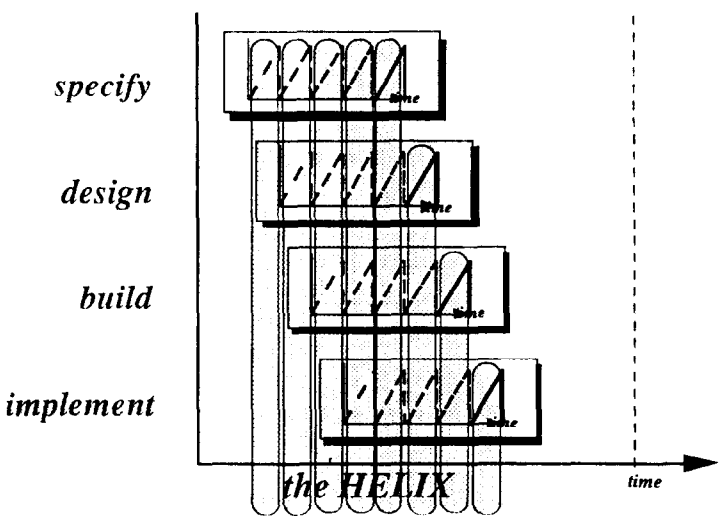

Figure 8 The limits of concurrent engineering

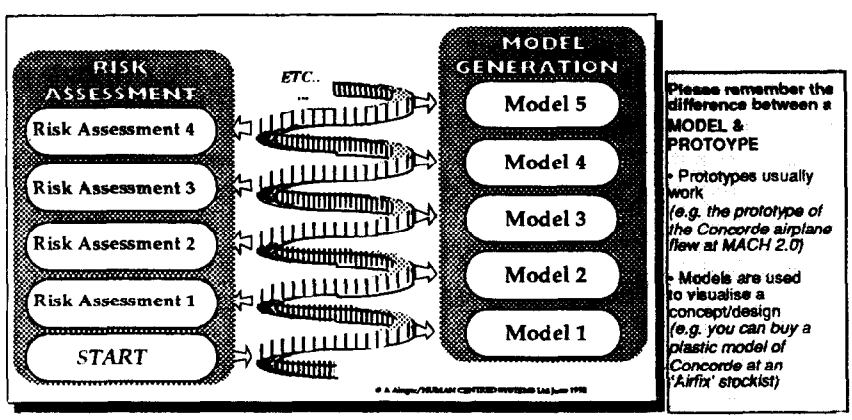

Figure 9 Outline of the helical project lifecycle design

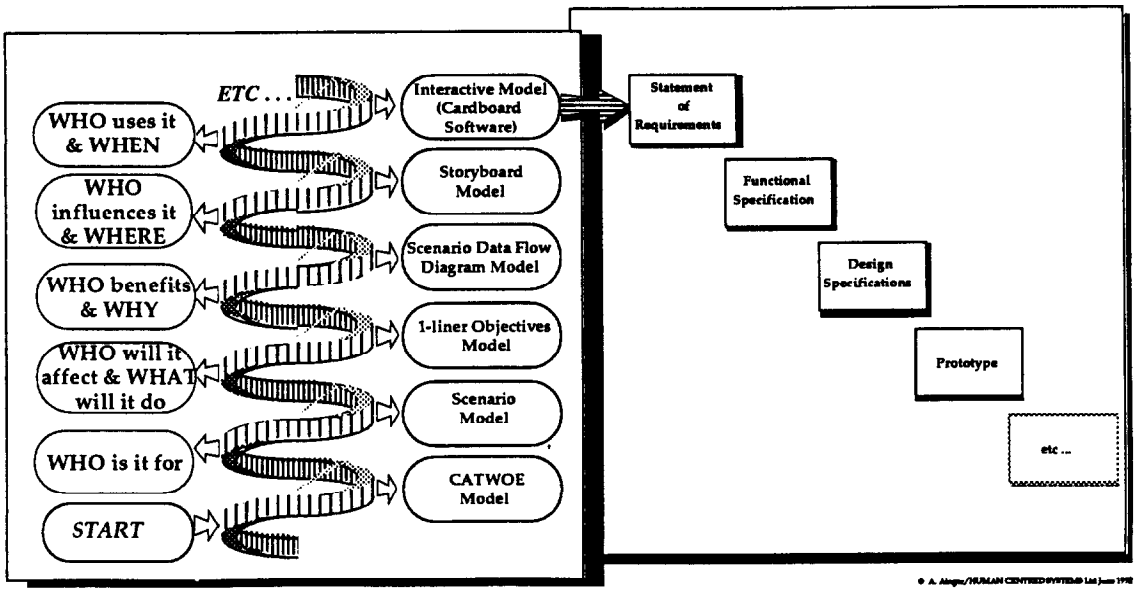

Figure 10 The Helical project lifecycle design approach

of architectural models. They provide a realistic view of the product at little cost.

For psychological reasons, it has proved useful, after providing the final interactive 'cardboard model' (*Figure 10*), to take the model and put it in a form of words (a process which can be likened to the creation of a requirements specification). This is not a necessary step as far as the method is concerned, however, we have found that many organizations, or rather people within those organizations, feel more comfortable with a 'requirements specification' as a document rather than a disk containing 'cardboard software'. A similar approach could be taken with the functional specification. This, also, is not strictly necessary, as all the major functions will be represented and displayed in the interactive cardboard model. However, we have found that some of the more traditional managers prefer the production of these paper-based specifications.

Should it be necessary to write down the requirements specification and the functional specification, it has been found that, after the production of the various Helical models, these two documents can be written extremely quickly and effectively, as it is far easier to write about something that can be seen (i.e. the 'cardboard model') than about something that cannot be seen (i.e. as would normally be the case). After all, a picture is said to be worth a thousand words. Provided the senior managers are introduced to the Helical method, the production of the systems specification and the functional specification documents have *not* proved necessary steps. What is necessary, however, is a detailed design specification.

It has been found that the 'cardboard software' produced can represent an ideal constraint rather than a realizable set of objectives. However, the risk of the 'ideal' solution being impractical, in software terms, is low, provided the system designer/model maker has sufficient software experience and sufficient discussions have occurred with the software authors. It is, therefore, for psychological reasons (comfort factor) that that Helical approach is shown to feed directly into the traditional waterfall lifecycle model. However, when the Helical approach is understood, and with a certain amount of practice, it is possible to use a 'life-shaft' inside the Helix and jump to the most appropriate model that the solution seems to suggest, then move to the interactive cardboard model, then straight to the design specification or even prototype.

Models and the helical approach

The Helical approach is focused around the generation of models. The quote the *Oxford Dictionary* (1990), a model is 'a *representation* of designed or actual object; design or style to be followed; give shape to, form'.

The following sections outline the various 'models' used in the Helical approach and, where appropriate, provides examples. When following the traditional waterfall approach, most feedback to the design stages occurs far too late. Real feedback generated by the users appears when the first prototype of the system is demonstrated. No matter how many specifications or documents are generated, it is only during the latter stages (i.e. the prototype stage) that real 'communication' takes place. The waterfall's single most positive aspect, feedback into the design specification by the users when viewing the prototype, has been seized and built upon in the Helical approach. The difference between a model and a prototype must be stressed at this juncture. In this context, models are used to represent/visualize concept/design, whereas a prototype actually works (i.e. the prototype of the Concord aeroplane actually flew at Mach 2).

In the Helical approach the generation of models assists both the user and the systems designer to communicate effectively. It is somewhat surprising to find that the use of models (as opposed to prototypes) is almost unheard of in the software field. However, in every other engineering discipline the generation of models can be a pre-requisite before the system/product/artifact is built. For example, if we take an aeronautical engineer or a shipbuilder or a civil engineer, each will build, or have built, a 'cardboard' model of the project prior to the build stage. It is almost certain that if one walks into an architect's office the first thing to be seen will be, under a glass dome, a 'cardboard model' of the latest mega project, a bridge, a dam or a block of flats. This model is not expected to work. The doors, windows and lifts in the model of the block of flats are not expected to be functional. What is expected is that the overall view, the general picture that is given is as accurate as possible[11].

It is relatively straightforward to conceive and build a model, whether it be in cardboard or any other material, of a block of flats. What is more problematic, though, is the construction of a 'cardboard model' of a proposed software product! However, just as one can have a model of a block of flats made of cardboard or other materials, it is also possible to have many models of the proposed software system. It is only towards the end of the model generation sequence that 'cardboard software' is used. Many other model making 'materials' have to be employed prior to the cardboard software stage.

The model making 'materials' of the Helix are not new. Some of the materials have existed for many years. The Helical Project Lifecycle Approach employs and utilizes concepts and insights from other disciplines to maximum benefit. The following sections give a brief outline of each of the individual models that go to make up the Helical Approach (*Figure 10*).

CATWOE model

The CATWOE Statement originates from the field of 'soft' systems theory. A version of it is used as the first 'Model' in the Helical approach and assists the users and system designers to come to an agreed, but short, definition (less than one page long!) of the system that is being developed. This convergence of ideas at an early stage of the project both intensifies discussion and focuses, as far as is possible at this stage, the minds of the people involved.

Checkland[19] defined the acronym CATWOE, where C stands for Customer or beneficiary, A = Actors in the system, T = Transformation, W = World image, O = Ownership, and E = Environmental constraints. The inputs and outputs also have to be identified, but the details of the CATWOE Statement can be read elsewhere (see Checklands original series of papers).

At a NATO Conference[17] a new acronym was suggested. Rather than CATWOE, it might be more appropriate to use COWPATÉS. It was felt that the current CATWOE acronym lacks two issues:

1. The purpose (P).
2. The softer issues (S).

At a number of project meetings subsequent to the NATO conference, it became apparent that the 'CATWOE Statements' that were suggested almost automatically included the purpose of the projection the first sentence. The softer issues or people issues were generally not mentioned. It is felt, therefore, that the acronym COWPATÉS is a more useful word to remember when using this particular model.

Scenario model

The next model on the Helix is the Scenario Model. The scenario has been used widely and successfully in the applied psychology field, and elsewhere, for many years. In building the Helical Scenario Model, the system designers and users, together, build a story/scenario around the problem/solution space*. It has been found that role playing the problem scenario can be a very exciting and revealing exercise. Once the problems have been identified, the solution role play can commence. The solution role play is normally not very stimulating as all matters should flow smoothly and all transactions should be completed successfully. However, the solution role play does focus, once again, both in the system designer's and user's minds, a mental image/model of the final system. For a partial example see *Figure 11*[16]. This example is taken from a multi-million pound Pan-European development project with partners in Italy, Portugal and the UK.

One-liner objectives model

When the mental models of both the designer and the system user are closely aligned it will be possible to write down the 'One-Liner Objectives Model'. In this task three aspects have to be considered:

1. Statement of a situation in which the future system is to be used.
2. Statement of management's expectations of the system.
3. The system's essential objectives.

These items are meant to be extremely brief. They are not meant to be all encompassing. What is intended is to fix, yet again, but in another way, the current state of both the system developers' and the users' understanding of their mental models. There should be, in general, no more than half a dozen or so principal objectives, and both the system designers and users should be able to agree, albeit in a general sense, on a form of words. A partial example is given in *Figure 12*.

Scenario data flow diagram model

The scenario data flow diagram (SDFD) combines, in the visual form, all the previous models (i.e. one-liner

*In one particular case in Italy, the solution scenario was only resolved by the various members in the meeting role playing, not only the users of the system but also the computer terminals themselves!

Operating Scenarios

The Consortium has prepared short written sales enquiry scenarios that briefly describe what is expected to happen when the sales group in a factory receive enquires for existing products, similar products and new productsThese scenarios help with the definition of the data required in each of these situations and also help to focus the Consortium's attention on the operation of the final integrated system.

Existing Products

The characteristics of this scenario are : the Sales Group are dealing with a customer enquiry for an existing product for which there will be a product numberprocess routes, part programs, and set-up and run-time data. It will therefore be possible for the Sales Group to establish if a requested delivery date is possible, or to establish a realistic delivery date using the sales enquiry handling DSS.
No interaction is needed with the design group. However, it is necessary to consider . . .

Similar Products

This scenario is concerned with . . .

Figure 11 Example scenario model

Statement of Objectives

1. Statement of the Situation in which the System is to Operate

The Sales DSS will be designed for a manufacturing environment characterised by the following features :

a) Cellular manufacturing environment manned by the cell teams with other 'natural groups' located in other parts of the factory (e.g. the sales team)
b) The cell and sales teams are involve in continuing improvement activities.
c) Manufacturing data are inaccurate and . . .

2. Essential Objectives

The Sales DSS should :

a) Allow more accurate delivery date to be given to the customer
b) Allow members of the teams to monitor trends in data accuracy and to identify causes of any improvement or deterioration.
c) Provide means of identifying and ignoring extreme values (rouge data) that are likely to distort the data analysis . . .

Figure 12 Example one-liner objectives

objectives model, scenario model and the CATWOE model). The SDFD is not a strictly data flow diagram, but they are built up of the same four symbols (*Figure 13*). In traditional data flow diagrams the time element is not well represented. In the scenario data flow diagrams the time element is represented by displaying a sequence of the diagrams one after the other, building up to a full scenario data flow diagram.

The scenario data flow diagrams (SDFD) are constructed by considering, in turn, each of the scenario models (*Figures 14a, b*). Each of the scenarios should be discussed in some detail, and a scenario data flow diagram constructed. The scenario data flow diagrams should then be combined to form a diagram which contains all the elements of the previous diagrams. It will then be possible to explain, using this one (composite) scenario data flow diagram, any element of the system/scenario (*Figure 14c*).

Storyboard model

The storyboard model is just that. The concept emanates from Walt Disney's production process,

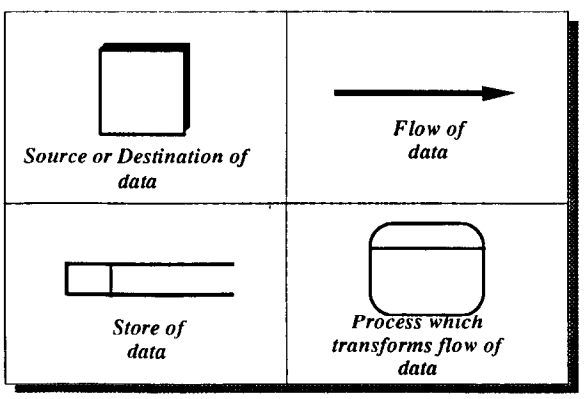

Figure 13 Diagrammatic nomenclature

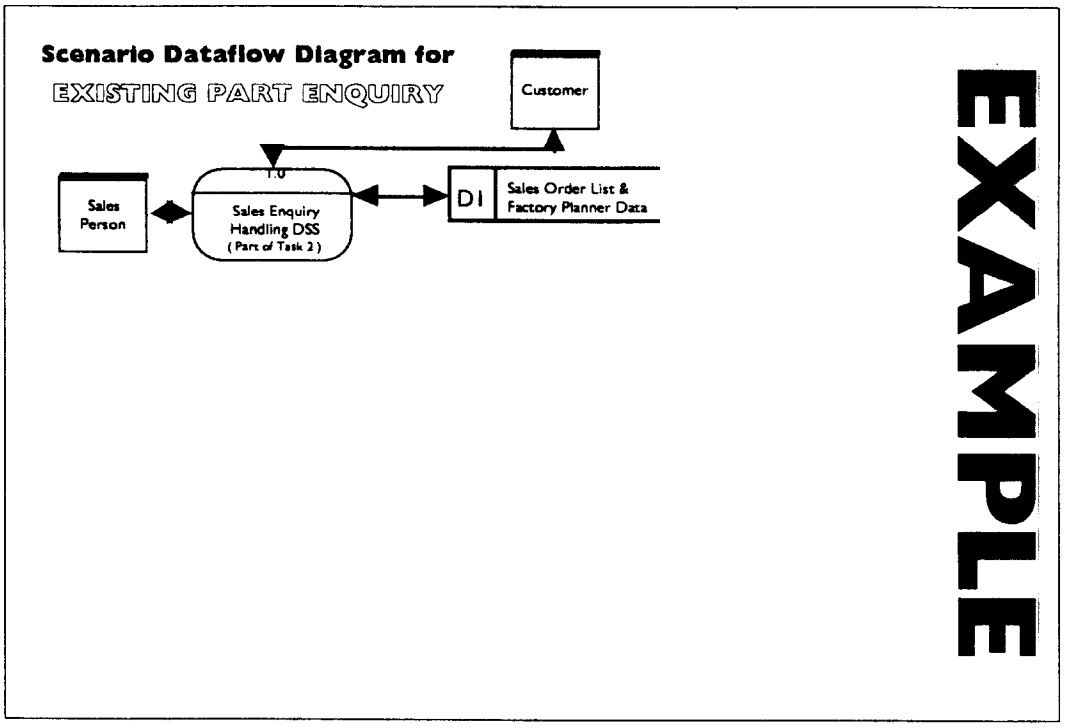

Figure 14(a) Example SDFDs. Existing part enquiry

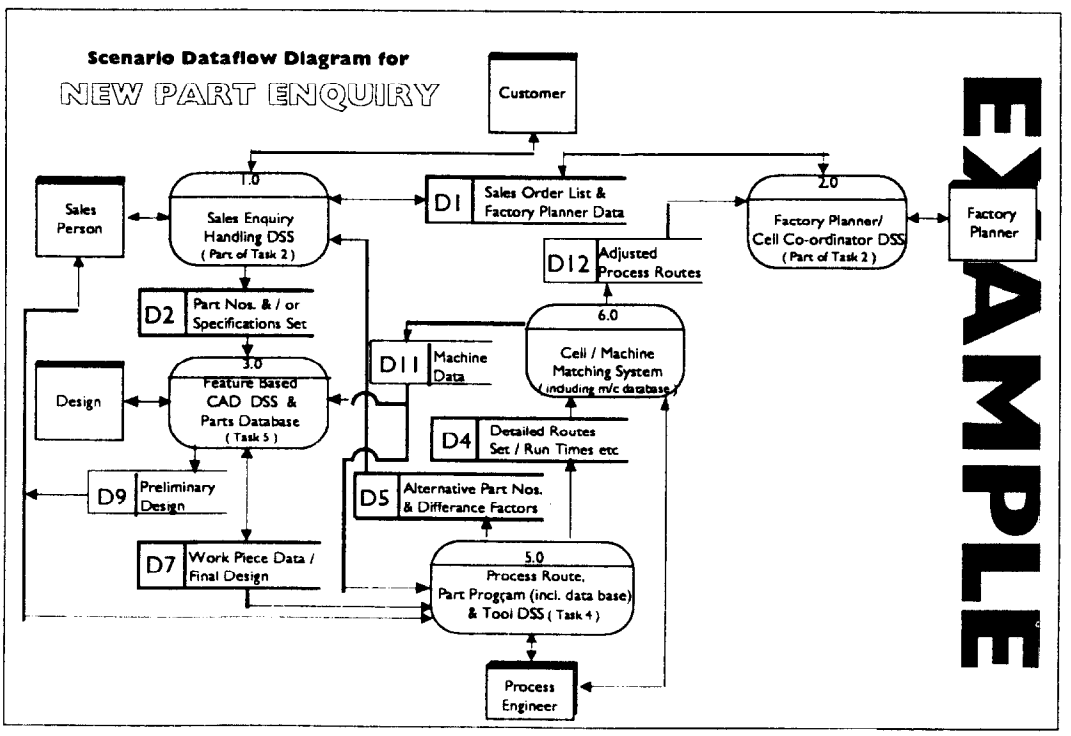

Figure 14(b) Example SDFDs. New part enquiry

where the principal characters and storyline are portrayed in sketches on pieces of paper or card. As far as the creation of a software system is concerned, the sketches that go to form the storyboard model should indicate the basic outline of the screen formats and should focus upon the information flow lines on the SDFD, surrounding the sources and sinks/destinations (i.e. the square boxes) of data.

Some connection should also be made to the time element. It is best if all these sketches are placed on a very large wall and the hierarchy of the screens portrayed such as to represent a basic time sequence. Various menu options can be shown as parallel tracks on the storyboard model. This is best explained via the case study (*Figures 15* and *16*). It has been found beneficial to focus on the principal output screens and

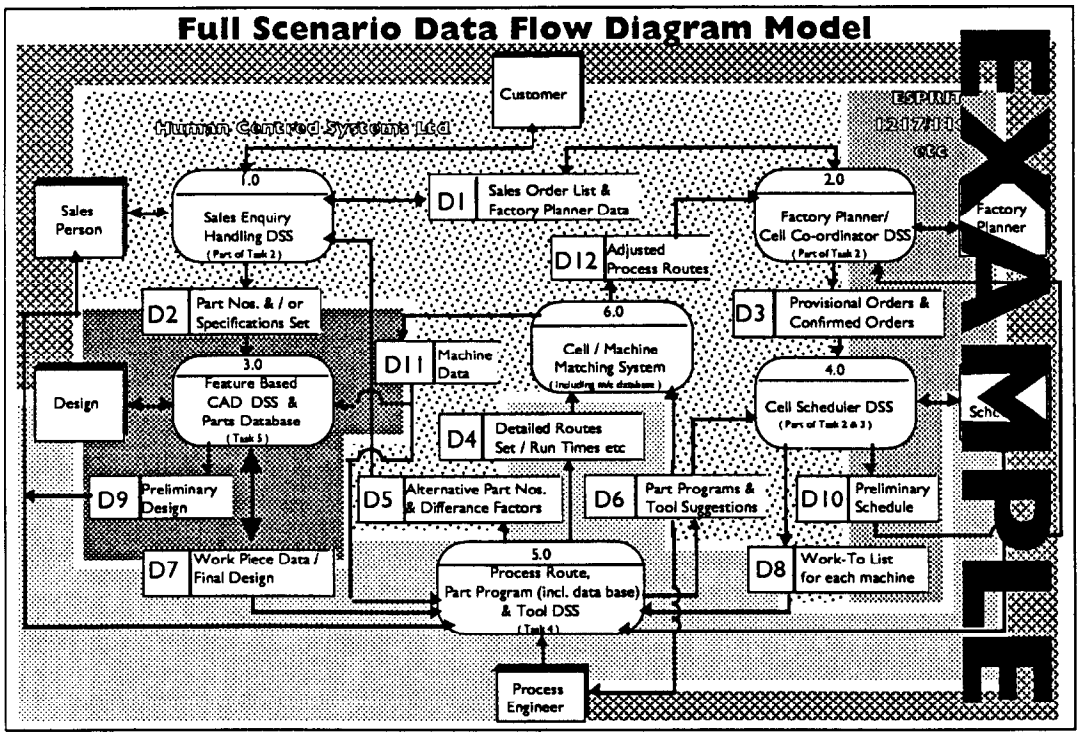

Figure 14(c) Example SDFDs

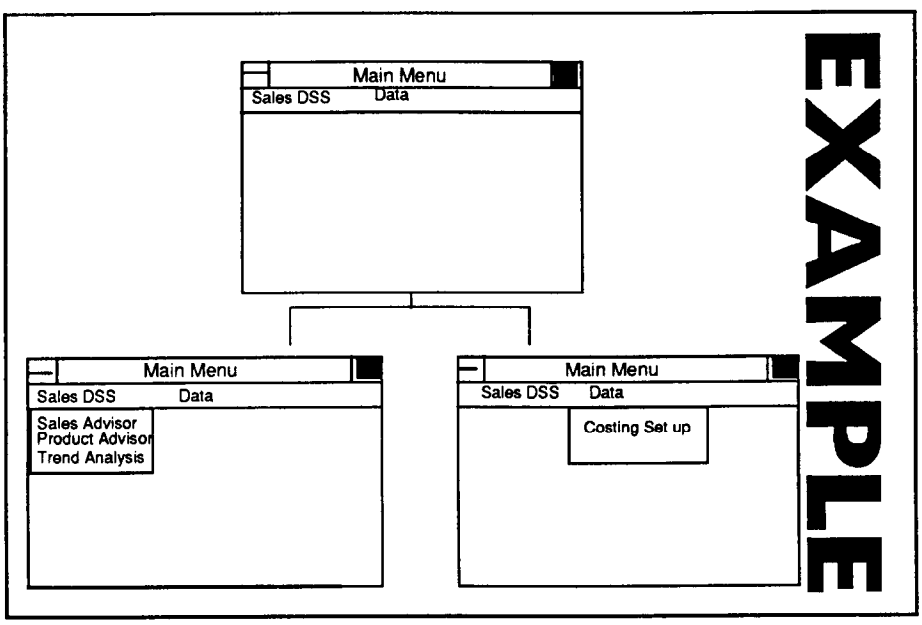

Figure 15 Example of storyboard model

the navigation between these. This should then be followed by the principal input screens. It is not necessary, or even desirable, to finish one before starting the other. The iterations between output, input and navigation issues all assist the designers and users alike to gain, share and jointly develop a common mental model of the proposed system.

Interactive model (cardboard software)

It is widely recognized that software products have a certain 'look and feel'. What has been established up to this stage is not only the outline functionality of the system, but also, through the storyboard models, the *look* of the system. The interactive model or cardboard software now starts to get a handle on the *feel* of the system. Creating the cardboard software from the storyboard model is relatively straightforward. There are many packages that permit the painting or drawing of screens. There are also a few packages that enable the linking of these screens in a pseudo-realistic manner[12]. Both the painting packages and linking packages are, nowadays, very cheap and are usable by non-software engineers!

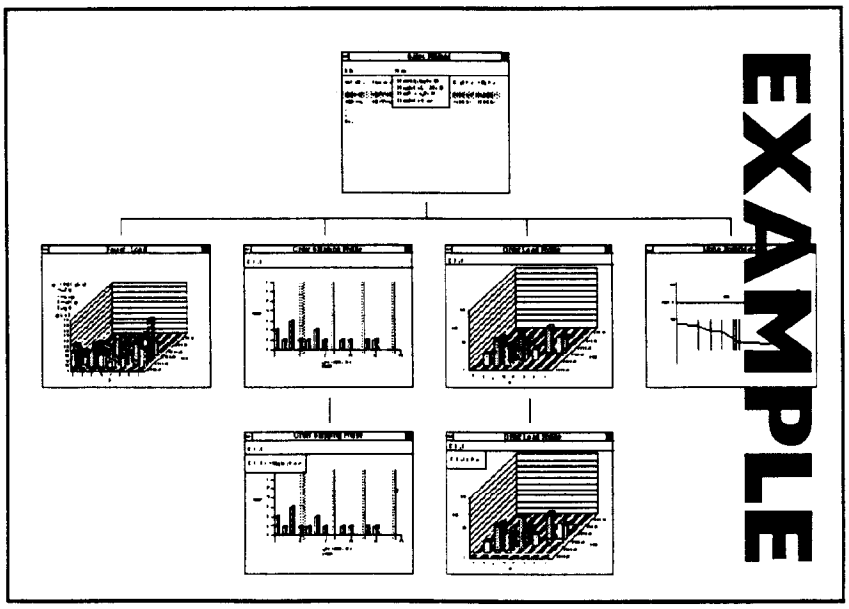

Figure 16 Storyboard model (detail)

It has been found that the 'look' and the 'feel' of the 'cardboard model' of the system can be most convincing. The model should be a realization of what is now a common mental model. It should represent a view that is shared by both the system's designer/developers and the user. However, even at this late stage, changes to the cardboard software can be made extremely quickly (a matter of minutes in some cases). This can represent significant savings in the overall systems development timescales and project finances, as any change made to the system when the product is demonstrated (i.e. in alpha, beta or even prototype form) are normally extremely expensive. Using 'cardboard software', it is possible for the users to scan through the system, view expected output, input and the navigation between the screens, and obtain an overview, a shared image of the final system.

Case study

The Helical Project Lifecycle has been developed over the past seven years, and has been used in many projects. The work was first reported in an ESPRIT meeting in Brussels during 1989[8] and developed further during 1990[9]. The Helical approach gained ground in 1992[10] when it featured in the UK's Department of Enterprise 'Usability Now!' campaign, and where a representative of the ITT Group stated 'In a period of rapid market change, effective user involvement in the design and implementation of our manufacturing process has been shown to be the key to success'. Whiting[3] takes it further, where he says that for the future 'product development is the key competitive battleground'.

The Helical approach has been used in several pan-European multi-million dollar projects, one of which was BRITE Project 3302[16]. This project consisted of five partners from three countries (UK, Italy and Portugal). The user site was a company that made moulds (up to 20 tonnes) for the plastic injection process. These moulds enable other companies to mass produce a variety of plastic products, principally in the automotive sector.

The project was focused into four areas. The examples below are primarily taken from the sales area. The aim of this part of the project was to develop a 'Decision support system for the salesperson'. An example, or partial example, of each of the above-mentioned Helical models is described. The resultant products are planned to be released at the end of 1995.

An example of a CATWOE Model is as in *Figure 17*[16]. It must be remembered that the CATWOE model is dynamic; it changes throughout the project to reflect current thinking. As a result, it must be up-dated and reviewed regularly. However, it summarizes the consortium's common understanding of the project's goals at that moment in time. *Figure 17* is one of the many CATWOE statements that were produced in the life of the project. It is interesting to note how these change over time and what appears to take dominance. For example, in the CATWOE development it was important to emphasize that 'a system' is not just a computer system, but involves people as well.

In the scenario model three areas were considered. First, it was assumed that the operating environment of the company concerned was known by all recipients of the scenario model. If this were not the case, a certain amount of confusion would arise. In the case of the Portuguese user company, three scenarios were discussed. Firstly, that of receiving orders/enquiries for products that had been made before; secondly, receiving orders/enquiries for products that were very similar to products that had been made before; and thirdly, receiving sales enquiries about completely new

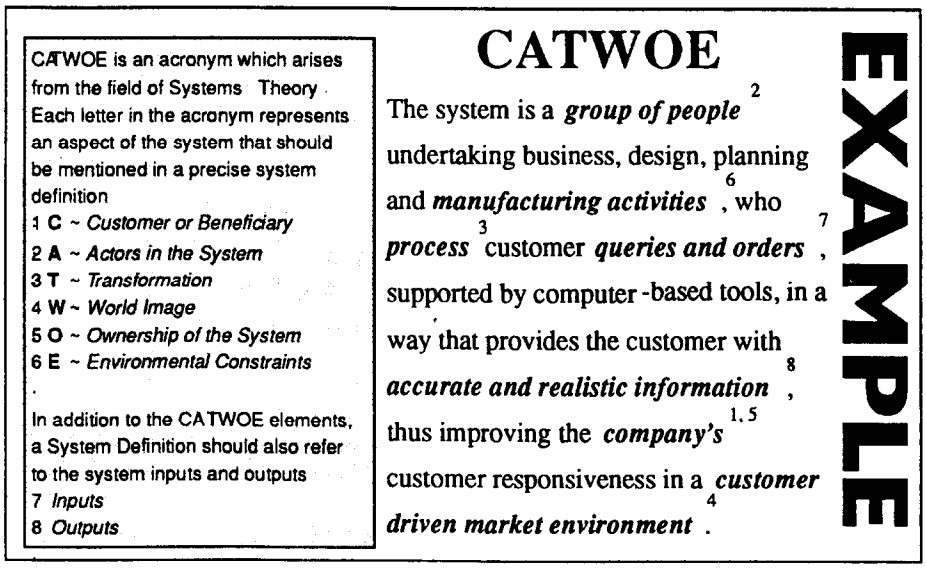

Figure 17 CATWOE example

products. It was felt, initially, that these three scenarios covered the full range of possible operating scenarios in the company concerned. However, after further detailed analysis (after the initial scenario data flow diagrams were produced), it became necessary to modify these in line with the revised needs of the users. *Figure 14* shows the results of this process.

In the modified sales decision support system, one scenario was to consider an order on a factory by an existing and known customer requesting an existing and previously supplied product (*Figure 14a*). Another scenario could be an unknown customer requesting a totally new product that has to be designed from scratch (*Figure 14b*). There are many other scenarios between these two extremes.

All the examples (i.e. *Figures 11, 12, 14–17*) used in this paper are taken from the same project[16].

Benefits

The benefits of adopting and using the Helical Project Lifecycle have been substantial. The Helical approach has been used on a number of projects, both internal to the group and externally. The approach has been used both nationally with the UK and internationallly on large pan-European software projects. Progress on projects employing the Helical approach have been rapid and success high[8,13,16]. To-date, the largest single project in which the Helix has been used successfully was a £5.6 million pan-European development project[7]. The first product (ACiT) to result from using the Helical approach was launched in London in 1990[14]. Major companies such as BICC and ITT Cannon have used the Helical approach, and all Human Centred Systems Ltd's (UK) products are now developed using the Helical approach.

The future

What does the future hold regarding the Helical approach? Obviously, the various forms of models and their applications can be improved. Lessons can be learnt not only from the fields of applied psychology, systems theory and structured analysis, but also from art and design, fashion and style houses, and other such sources. However, we believe the major change that should come fairly quickly will be in the IT departments that utilize the Helix. It is becoming apparent that a new job function has to be created, that of model maker!

Model makers have existed in the traditional engineering sector for decades. For example, who makes the cardboard model version of the block of flats? Obviously, the bricklayers involved on the building site would not be a first choice. However, for some reason, when considering software models, most organizations tend to use software people (the equivalent of bricklayers in the block of flats!). What is required is a new engineer: a software model maker. An engineer who is a true hybrid; a person who understands not only the limits and boundaries within the software engineering field, but also someone who has a grasp of psychological aspects; the ergonomics of screen design and an understanding of style; layout; form and function.

We are not suggesting that these new software model makers be versed in all the details of software or any other particular discipline, just that they have a broad understanding of a number of engineering issues. These people need not necessarily write the real software – just as the real model makers do not make the real block of flats. These new software model makers must have experience in writing software products, but what is more important is that they must have a firm understanding of social and psychological issues and a flair for art and design and, most importantly, be expert communicators (not necessarily verbal).

In the past, when computer systems were in their infancy, this level of specification sophistication was not necessary. The systems could be changed and re-written relatively quickly. However, in today's complex software arena the software profession is gaining

maturity, and it is now time that we adopt appropriate and relevant procedures and practices, which are commonplace in more traditional engineering disciplines and learn from their experience.

Conclusion

The Helical approach to software design is a re-assembly of pieces from old jigsaws. The pieces, when assembled in this novel way, make an exciting new picture for the future. The speed by which software systems can now be developed is improving all the time, and the accuracy by which they reflect user requirements can now be made more certain. The benefits of using the Helix can be startling both in small projects and large ones. The time and financial savings that can accrue by adopting the Helical approach can be beneficial not only at a group level, but also at an individual level. The biggest problems found when implementing the Helix have been the mental barriers within the individuals concerned. It is almost second nature to go from specification to design and design to prototype. What we have to do is to break down these mental barriers. We have to think in a Helical way and to build models, to become multi-skilled, to think globally and to learn continually whilst we (hopefully) earn.

References

1 **USA Department of Defence** GAO (1979)
2 **Ernst & Young** *'STAGES Methodology'*, Ernst & Young, London, UK (1991)
3 **Whiting, R** 'Product development as a process', *Elect. Bus.*, Vol 17 No 12 (1991)
4 **Finkelstein, L** 'The life-cycle of engineering products – an analysis of concepts', *Eng. Manage. J.*, Vol 1 No 3 (1991)
5 **Butler, M** 'Too much order can mean chaos', *Computing* (1992)
6 **IEE News** 'Making IT work for users' (January 1991)
7 **Hamlin, M** 'Human-centred CIM', *Professional Eng.* (April 1989)
8 **Ainger, A** 'Human-centred design of human-centred systems', *ESPRIT Conf.*, Brussels, Belgium (1989)
9 **Ainger, A** 'Aspects of an experimental industrial application of human centred CIM (HC-CIM) systems' IEE Colloquium London, UK (1990)
10 **Ainger, A** 'How human factors expertise can improve manufacturing systems', *Usable IT in Manuf. Conf.*, Birmingham, UK (1992)
11 **Ainger, A** 'If only', *Electronics and Wireless World* (January 1992)
12 **SEEiT** *Screen Linking Product*. Human Centred Systems Limited, Old Windsor, Berks SL4 2JP, UK (1993)
13 **Ainger, A** 'Manufacturing – a practical human-centred perspective', *Eng. Manage. J.* (1991)
14 **ACiT** *Factory Loading and Cell Sequencing Software Modules*, IESL, Manchester, UK
15 **Usability Now!** *The Enterprise Initiative*. Department of Trade & Industry, Scalectro Case Study (1991)
16 **BRITE Project No. 3302** EEC, Brussels (or for further information contact Human Centred Systems Limited, UK. +44 (0)1753 833557)
17 **NATO Conference** 'People and Computers', Loughborough University, UK (1993)
18 **Schmidt, F *et al.*** *Computer Integrated Production Systems and Organisations*, NATO ASI Series F, Vol 134 (1994)
19 **Checkland, P** *Systems Thinking, Systems Practice*, Wiley, Chichester (1991)