Chapter 3: Review of literature

Introduction

In this chapter the various well-established techniques from the world of manufacturing, knowledge management, knowledge based design, Case-Based Reasoning, objects, Kansei Engineering, Quality Function Deployment and TRIZ are analysed in depth in an attempt to discover if improvements can be made to the early phases of design.

It is not intended to criticise the product innovation methodologies discussed below and mapped out in Figure 2. The intention is rather to establish what value these methodologies could add to the initial and also subsequent stages of the design process. At this stage it is also presumed that the eventual solution proposed in this study should be such that any external product innovation methodology (application) can be applied to it whenever desired.

It is beyond the purposes of this study to provide a detailed discussion of the numerous different design theories.

Over the years many different product development techniques evolved in the world. None of these product innovation methodologies or techniques is capable of solving all the problems inherent in the product design process. However the rich set of techniques is successful in solving many of the sub-aspects.

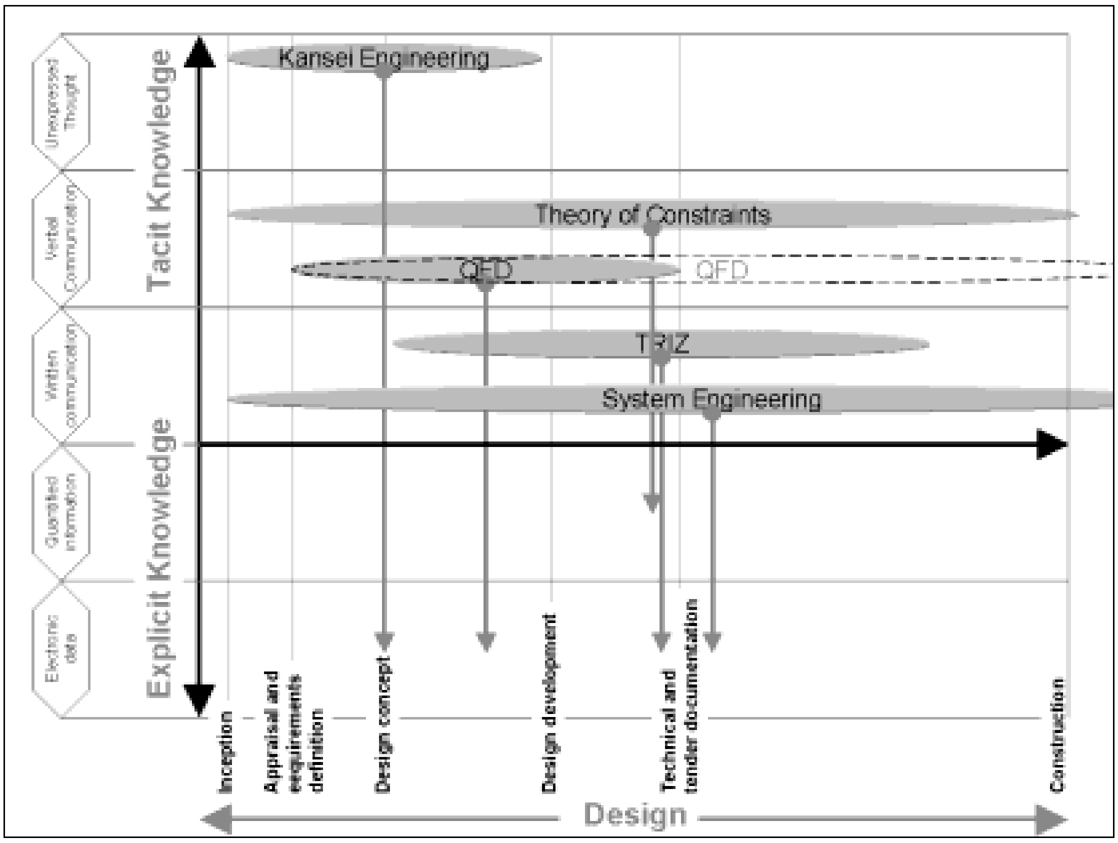


Figure 1: Product innovation methodologies (Collated by author)

The author is of the opinion that the great difficulty that is experienced is partly due to the fact that technology, especially electronic computing, exploded beyond all recognition. This has the effect that most problem solvers are almost by default attempting to solve the particular problems by means of computer models, rules or cases. This creates the incredibly difficult problem that design knowledge must first be quantified or moved from the tacit level to the explicit level (Figure 1) to be in a readily processable form on the electronic computer.

The following current techniques exist that could assist the product development team in product innovation (Figure 1). Although many more techniques exist the present study concentrates on *Case-Based Reasoning* against the background of *Theory of Constraints*, *QFD*, *TRIZ* and *System Engineering*. The assumption with sub-problem 1 was that techniques from the manufacturing industry would only partially solve the problem and that a bridging technique between the human brain and systematic (structured) approaches will have to be established. CBR is specifically studied in the light of sub-problem 3 to see whether designs can use experience from the past to expedite present designs.

Figure 1 maps the different tacit levels where these techniques source information and indicate how far the information can be quantified or moved to a level of explicit knowledge. On the horizontal axis the various main stages of the architectural design process are mapped out. Traditionally QFD was only really useful in the initial design stages up to design development. However in the prototype system *AEDES* attempts were made to stretch it further across the life cycle of the structure, hence the dotted line. *Natural Language Processing* (NLP) is an evolving technology that could greatly assist to turn the spoken and written word into explicit knowledge. Attempts are currently been made to develop NLP software to facilitate concept extraction out of text. However it is going to take at least another three years before a sufficient level of reliability is reached.

The accurate extraction of customer needs are very important for ultimate success of the architectural design. Some methods for defining customer needs are (Zultner 1999):

- *Kansei Engineering* – emotions of the customer.
- *QFD* – voice of the customer.
- *Theory of Constraints* – mind of the customer.
- *Customer Context Analysis* – context of the customer.
- *Systems Dynamics* – environment of the customer.

Methods for structuring product acquisition

- *Business Engineering* – attempt to structure entire product environment.
- *Systems Engineering* – structured methodology for sequencing design activities.

In order to create a digital representation of an architectural design artefact it is necessary to create a building product model. A building product model is a digital information structure of the objects making up a building. It captures the form, behaviour and relations of the parts and assemblies within the building. Major efforts are being made throughout the world to develop such a representation. Among the industry groups involved in achieving this are the U.S.A. based Product Data Exchange using STEP (PDES) organisation and its international counterpart, International Standards organisation – Standard for the Exchange of Product model data (ISO-STEP). Efforts are also being undertaken by research groups funded by the European Union in Europe and by the National Science Foundation in the U.S.A. Another significant effort is of the International Alliance for Interoperability (IAI). According to Eastman (1999) building product models will eventually be used by most people associated with the building and real estate businesses such as architects, engineers, contractors, owners and facility managers.

In this study the use of the structured hierarchical ASCII standard, XML will be used to explore the creation of a simple, yet powerful design language to support desk top based design tools such as design modellers whilst at the same time maintaining a close relationship with the Internet. The opinion is expressed that the ASCII nature of XML makes it a strong candidate for an application independent design language because it is simple but at the same time flexible and extendable. It is also a very powerful integrator of non-XML data. XML provides three constructs that could be used to achieve flexibility and extendibility:

- Notations
- Unparsed external entities
- Processing instructions.

3.1 Knowledge management

3.1.1 Introduction

This section is included because architectural design depends to a large extent on the availability of sound knowledge. Some of the wide range of knowledge that is required in this domain can be summarised as:

- National Building Regulations.
- Characteristics of materials.
- Construction components.
- Construction methods.
- Energy use.
- Surveyor General’s diagram.
- Acoustics.
- Solar movement.
- Anthropometrics.
- Climatic conditions such as temperature, wind and rainfall.
- Construction detail.

If this type of information cannot be readily obtained then it makes the designer ineffective. It is highly desirable that the designer is able to quickly retrieve this knowledge whenever required. Whatever solution is ultimately proposed its success will depend to a large extent on the convenient access to this type of information. With the advent of large construction projects it is also crucial that designers are able to collaborate globally. Microsoft’s approach to KM is studied to establish if it is suitable for the proposed solution.

Nonaka (1998:22) states that in an economy where the only certainty is uncertainty, the one source of lasting competitive advantage is knowledge. When markets shift, technologies proliferate, competitors multiply and products become obsolete, successful companies are those that consistently create new knowledge. These companies are also able to disseminate it widely throughout the organisation and quickly embody it in new technologies and products. The creation of knowledge is seen in the context of the Japanese approach that creating new knowledge is not simply a matter of processing objective information. It depends on tapping the tacit and often subjective insights, intuitions and guesses of individual employees. These insights are then made available for testing and use by the company as a whole. Making personal knowledge available to others is the central activity of the knowledge-creating company, whose sole business is continuous innovation. By this is understood that According to Nonaka the following four basic patterns for creating knowledge in an organisation exists (Figure 3):

1. Tacit to tacit.
2. Explicit to explicit.
3. Tacit to explicit.
4. Explicit to tacit.

Harari (1999) confirms this with his interpretation of knowledge when he suggests that companies should have cutting-edge skills, state-of-the-art tools, creative tools, creative freedom, business accountability for employees and speed and intelligence in everything. This is all aimed at doing something truly special that amazes customers.

Tom Davenport defines knowledge as a fluid mix of framed experience, values, contextual information and expert insight that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of knowers. In organisations, it often becomes embedded not only in documents or repositories, but also in organisational routines, processes, practices and norms.

Knowledge management (KM), as defined by the GartnerGroup, is a discipline with new processes and technologies that differentiate it from information management. New technologies are required to capture knowledge that was previously tacit. Tacit knowledge is embodied in the minds and expertise of individuals. Once captured, knowledge must be shared to leverage its value and reused in similar situations and contexts.

The unique requirements of KM have inspired many startup ventures and innovations by the information industry (Figure 9). The needs for new technologies and the technological advances have occurred simultaneously. The need is to quantify knowledge, codified in digital form in the form of documents and apply it in new situations. The codified knowledge must be found first and searching is dependent upon natural language. New KM capabilities must overcome the ambiguity and context-dependent nature of natural language.

3.1.2 The nature of knowledge

Table 1: Stages of technological knowledge (Bohn 1997:77)

Stage	**Name**	**Comment**	**Typical form of knowledge**
1	Complete ignorance		Nowhere
2	Awareness	Pure art	Tacit
3	Measure	Pre-technical	Written
4	Control of the mean	Scientific method feasible	Written and embodied in hardware
5	Process capability	Local recipe	Hardware and operating manual
6	Process characterisation	Fine-tune the process to reduce costs	Empirical equations (numerical)
7	Know why	Science	Scientific formulas and algorithms
8	Complete knowledge	Nirvana	

Bohn (1997) identified eight stages of technological knowledge ranging from complete ignorance to complete knowledge (Table 1).

The stages that a field of endeavour goes through before it can be considered a science are (Goldratt 1990):

1. Classification
2. Correlation (The question *why* is not asked. *How* is at the centre of interest)
3. Effect-Cause-Effect (Know *why*)

The first stage, classification, in Facilities Management (FM) started about ten years ago. Numerous classifications were developed to assist in the various activities that are undertaken by facility managers. An example of this is the SAPOA standards with regards properties to calculate rentable and usable space in buildings. During the National Health Facilities Audit the concept of departments such as outpatients, operating and administrative were used to group spaces that share a common planning unit. This was used specifically to facilitate a large-scale condition and suitability analysis.

The second stage was entered about five years ago when FM developers and users realised that all the various operational and strategic FM actions need to be integrated into a holistic systems environment. The most important question at this stage is *how*. A characteristic of this stage is many different highly detailed models and approaches, but a lack of deeper scientific understanding. Both architectural briefing and design and FM are at the moment in this stage. The latter developed significantly faster than the former. The author is of the opinion that the scarcity of financial resources, energy and a general realisation that there are limits to growth (Meadows *et al*. 1975) provided the impetus. The domain of FM is far less challenging than architectural briefing and design, albeit extremely important.

The challenge of the present study is to make the quantum leap into the third stage where the *know why* becomes the characteristic. A complete mastery of *know why* would indicate that the level of science has finally been reached. Sir Isaac Newton finally turned physics into a science when he discovered the fundamental laws of motion and gravity. He asked the question why do apples fall down rather than flying in all directions? He assumed a cause for this phenomenon. He assumed the gravitational law. Explanation appears on the stage. It is a foreign word in the classification and correlation worlds where the only proof is in the pudding. Past experience is no longer the only tool. He published works such as Philosophiae Naturalis Principia Mathematica in 1687.

Nonaka (1998) clearly indicates that the classic pyramid model that consists of discrete layers of data, information, knowledge and wisdom is an oversimplification, because this model assumes a quantification of all data, hence a total transfer from tacit to implicit (Figure 2).

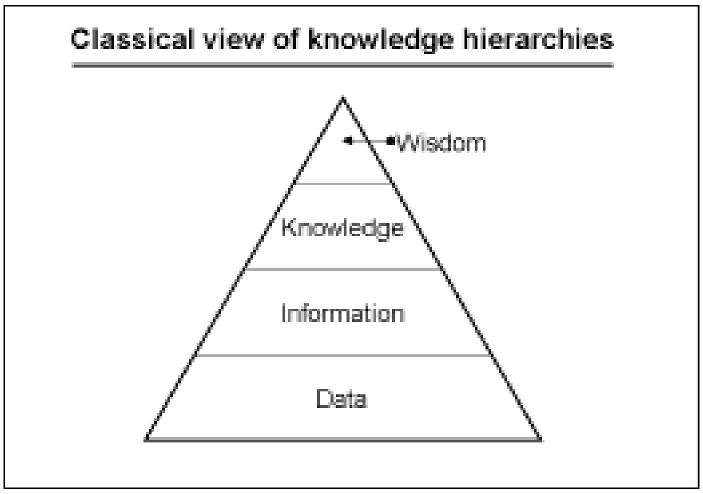


Figure 2: The classical view of knowledge hierarchies (Author)

Today it is recognised that the steps of the knowledge cycle consists of the actions of *create*, *capture*, *organise*, *access* and *use*. The most appropriate model of the knowledge cycle is by Nonaka (GartnerGroup 1998:2) and (Nonaka 1998:21). Humans have certain capabilities that lead to the four broad, fundamental human behaviours illustrated in Figure 3.

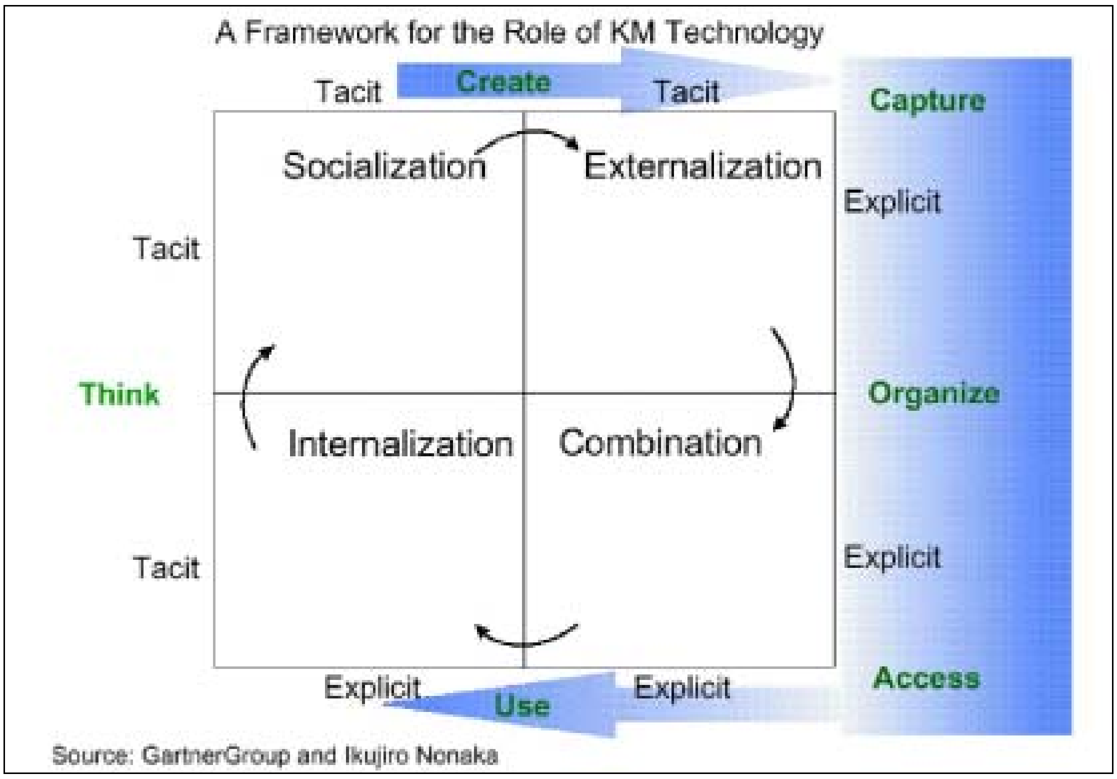


Figure 3: The knowledge cycle and process. See text. (GartnerGroup 1998:2)

3.1.2.1 Socialisation

This is people learning from each other by doing. Tacit knowledge is exchanged, as a by-product of collaborating, like an apprenticeship. Socialisation arises out of real-time connections among people to consolidate their knowledge. Currently, tacit-to-tacit sharing requires physical presence, but technologies such as virtual reality could eventually simulate presence.

3.1.2.2 Externalization

The situation where knowledgeable people consciously convert their tacit knowledge to an explicit form is called externalization. Publishing captures information, but KM requires capturing processes, conditions, rules, timing and other factors that can be re-created in subsequent situations. It requires that tacit knowledge such as a person's experience be engineered into an explicit form. Before KM, information was captured in documents, papers, E-mail and notes. These textual sources look like long strings of words to most software. KM technology attempts to represent the traditional content and new media by exploiting natural language processing (NLP), sets of rules, document structure, context and relationships so that it can be reapplied in related situations.

3.1.2.3 Combination

The activity of gathering and integration of the captured knowledge of individuals and groups for access by the enterprise is called combination. Combination requires that the engineered knowledge be evaluated, transferred to other groups and communities and leveraged through reuse.

3.1.2.4 Internalization

Internalization is the “experience” of the explicit knowledge by individuals, who learn by making the explicit knowledge their internal knowledge. Internalization employs the techniques of pedagogy to teach knowledge that is customized and applicable to individual needs. It requires engineering knowledge to engage the attention of the user.

3.1.3 The current situation

Currently tacit to tacit sharing requires physical presence, but technologies such as virtual reality will eventually simulate presence. Current KM implementations depend on a small subset of the types of products that are needed for the complete knowledge cycle. GroupWare and information searches are the most used with more advanced technologies appearing in less than 15 percent of implementations. The need to externalise tacit knowledge will drive new technologies.

The ability to access stored information is far ahead of the ability to find relevant information. Turning explicit knowledge into tacit knowledge is still the purview of computer-based training (CBT) and teaching. Developments in distance learning are starting to appear from vendors such as IBM/ Lotus. The use of products for meetings, such as group decision support systems, has had very limited success (GartnerGroup 1998). Physical and virtual workspaces are still experimental and tele-technologies are today less effective than face-to-face meetings.

Improvements are expected to be made over the next five years, but the majority of sharing will continue to be through oral communication for the next 10 years or more. There are four main reasons why KM technologies will remain tools rather than replace current behaviours in the knowledge cycle:

Situation. Each situation that requires knowledge is unique. It is not feasible to predict exactly what knowledge is required beforehand.

Language. Humans share what they know through the imperfect vehicle of language. Although language is imperfect it conveys precise information within the context of use that the human brain is well adapted to. Language skills vary widely. Language is often insufficient to represent other factors that cannot be expressed verbally. This is one of the reasons why more than 50 percent of knowledge worker communication is still face-to-face, where complex things like trust can be established.

Context. Making knowledge explicit often leads to storage out of context. The preservation of context is very important for architectural design, because designs are always solutions within the context of numerous requirements and constraints. One of the innovative methods to store or communicate contextual knowledge is by means of business stories or novels. It is not surprising if it is considered that the oldest means of storing information is found in epics such as Homer’s Iliad and the Finnish Kalevala. An outstanding modern example of this is the method that Goldratt, (1993) uses to explain the ideas which underlie the Theory of Constraints (TOC). In books such as “The Goal” he uses the context of a novel to explain manufacturing processes in the context of a manufacturing plant. The technique of using a novel is successful because it gives contextual meaning to the various powerful and innovative concepts explained. Because of the way that the human brain operates this method of externalization makes the later internalization of vast amounts of knowledge feasible.

By considering context while processing natural language it is possible to interpret the meaning of a sentence from related text by placing the individual sentence in context. According to Popov (1982) there are various levels of context that need to be considered when processing natural language.

Textual context is the meaning derived from the sentences preceding the current sentence.

Situational context is the meaning from the current sentence and is usually only given implicitly.

Global context is like the topic of a conversation and allows an algorithm to choose between several meanings. An example is the word overloaded that could mean having too much luggage, put to great a demand on an electrical system or the technical term in computer programming where one programming keyword such as "=" might have different meanings in different syntactical constructs.

Local context is the meaning derived from only the few preceding sentences. This is useful because the topic of the conversation may progress. Local context provides the most recent topic.

A simple algorithm for processing context and reference is not possible since a form of fuzzy processing is required. Researchers are experimenting with neural networks to train a computer to recognize certain common situations (frames) and to generalize about new situations.

A machine that processes natural language must be able to categorize, understand and process the wide variety of language components. Some of the different hierarchical syntactical parts of language identified by Russian analyst (Zvegintsev 1976) are:

- Discourse
- Sentences
- Phrases
- Words
- Morphemes
- Syllables
- Phonemes
- Differentiating signs

Relevance. The value of information is subjective. For information to be knowledge, it must get the user's attention relative to other information, comparable to a teacher-student interaction.

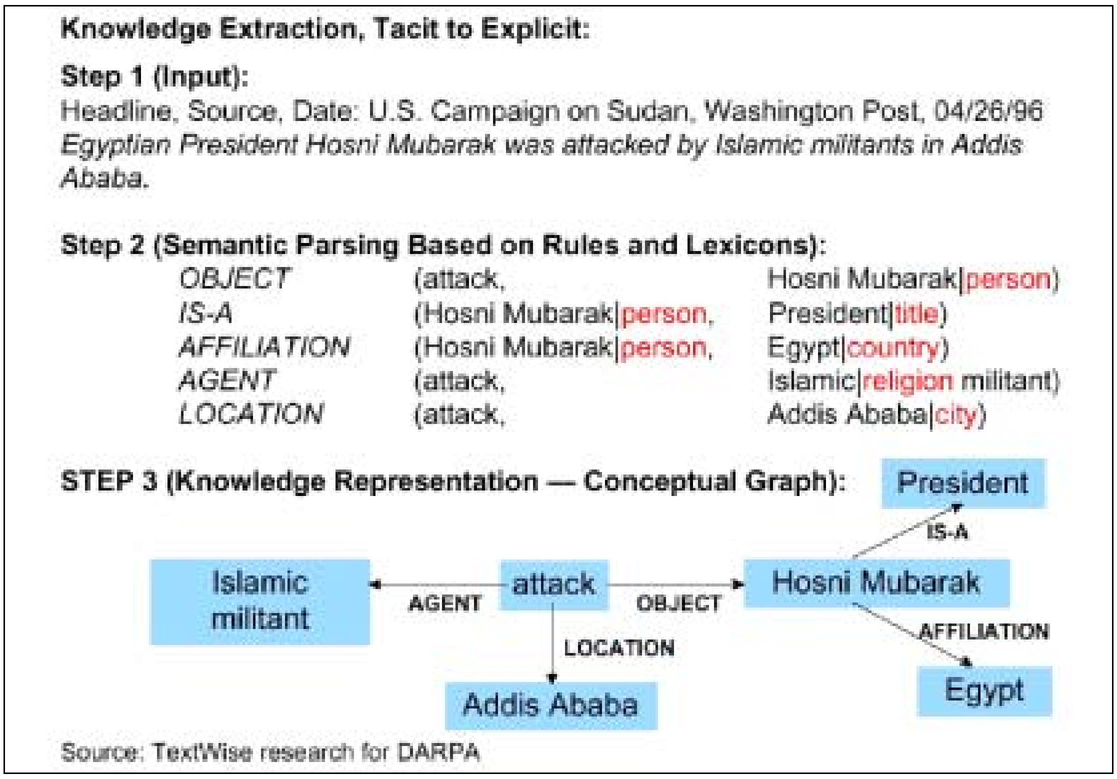
The representation of tacit knowledge is currently the focus of development, but it is unlikely that it will be used practically until after the year 2002 (GartnerGroup 1998).

3.1.3.1 Desirable emerging technologies to enable knowledge management

The storage and retrieval of knowledge in written form was viable with paper and limited online access before the advent of the World Wide Web. The present unprecedented quantities of information online made the retrieval of documents or records inadequate. New capabilities are addressing the conversion of text into more usable forms. It is now important to achieve summarization in order to find relevant knowledge. This is a capability that would increase the quality of stored knowledge. However, it depends on accurately representing the meaning of text. With the current information overload, users will want to know what a document is about rather than having to read it or a summary. Capabilities are being developed that describe the people, places and things discussed in document. This is at a far higher level than a pure keyword search. Users can search in a natural language way for subjects such as "Tell me about Microsoft, Internet components and Visual Basic". Unlike present search technology that search for those words specifically, representation technology will identify what Internet components are and respond with the latest Microsoft Internet Information Server technology used. The barriers to computer understanding of language results from the innumerable ways to express the same thing, the different meanings of words and the inherent imprecise nature of language structure. If information technology fails to achieve significant breakthroughs in representation technology soon the knowledge community will be faced with vast but virtually unusable knowledge stores.

If intelligent processing of design information is to take place on the electronic desktop then relevant structured information needs to be delivered to the electronic design tools. This indicates the necessity for a flexible and self-describing design language.

Concept extraction is at the same time one of the most promising and problematic of the emerging capabilities. It depends on NLP to parse sentences according to rules (Figure 4). In this particular example the Xerox Inxight Hyperbolic Tree was used in the Syracuse University TextWise system. *President* is capitalized and directly precedes a name. The lexicon says *President* is a title and *Mubarak* is a name. In a newspaper article users want to know how concepts are used and not just the fact that they occur. In a document that contains the concept "Chrysler" it could be an advertisement, stories about the company's stock performance or an article about a recent merger.



Knowledge Extraction, Tacit to Explicit:

Step 1 (Input):

Headline, Source, Date: U.S. Campaign on Sudan, Washington Post, 04/26/96
Egyptian President Hosni Mubarak was attacked by Islamic militants in Addis Ababa.

Step 2 (Semantic Parsing Based on Rules and Lexicons):

OBJECT	(attack,	Hosni Mubarak|person)
IS-A	(Hosni Mubarak|person,	President|title)
AFFILIATION	(Hosni Mubarak|person,	Egypt|country)
AGENT	(attack,	Islamic|religion militant)
LOCATION	(attack,	Addis Ababa|city)

STEP 3 (Knowledge Representation — Conceptual Graph):

Source: TextWise research for DARPA

Figure 4: Concept extraction for representation (GartnerGroup 1998:8)

By representing the concepts according to conceptual relations, the technology could infer the difference. Once concepts are extracted, they can be stored in databases according to their usage and relationships Concepts are also the basis of visualization capabilities. They are displayed on a screen where the relative proximity of each concept reflects relative similarity of meaning. This provides the user with a conceptual map of a knowledge domain, enabling navigation to the units of stored knowledge. It also shows relationships such as affiliation. For the next two years automatic linguistic representations will require access to manual checking through visual user interfaces to expose inaccuracies.

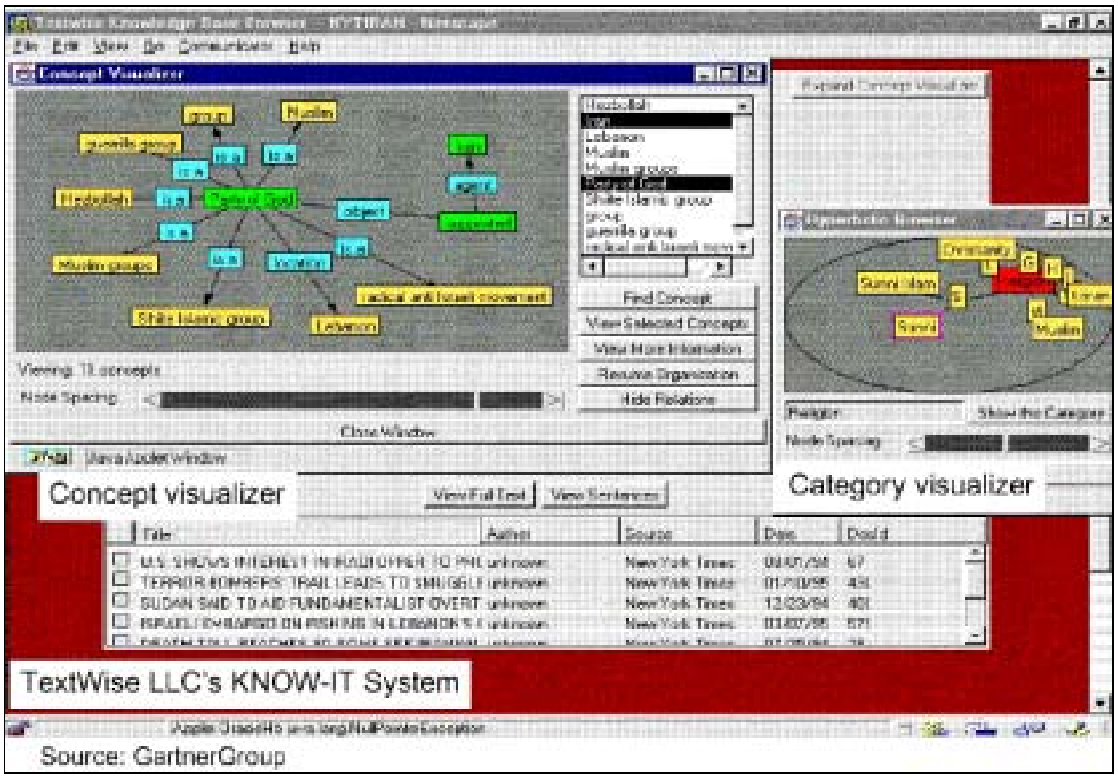


Figure 5: Visualisation of representation (GartnerGroup 1998:9)

Knowledge representation will expand beyond linguistics to include technologies that will converge over the next five years. Significant developments in collaborative filtering resulted in successful products such as grapeVine, NetPerceptions, Firefly and WiseWire. Users with similar profiles are considered sources of recommendations to each other. Profiles use representation technology to capture similar interests.

Concept extraction provides a way to describe documents. The concepts can be used as attributes in a database. Expert systems continue to be focused on narrow domains, which can be described by rules.

Knowledge representation is the conversion of captured knowledge into a reusable form. But it focuses on the tacit-to-explicit part of the cycle. It must be integrated with sharing media to overcome the scalability issues of space and time. Sharing over time has both a historical and a coordination dimension. Knowledge must be maintained over time or it will lose its value. It must be current, as is true of any information. Knowledge workers try to interact synchronously, in real time, because they want the latest knowledge applied to the current situation. Scheduling a simultaneous interaction is increasingly difficult, especially with worldwide enterprises. The two notions of time exacerbate each other. One of the largest demands in KM is to overcome the need to interact in real time to share current knowledge. The reusability test requires that person *A* must be able to make sense of and apply person *B*'s knowledge at person *A*'s time and place. When this is not feasible, users must resort to sharing media by means of E-mail and teleconferencing.

3.1.4 Knowledge management architectures

There are several barriers to a next-generation responsive information system, which can be rapidly composed or adapt itself to a new environment. Presently vendors are reluctant to support truly open architectures. Neither the vendors nor the users understand the importance or the nature of the interface between modules. In any modular system, the interface definition is critical to the assembly of those modules into larger functional modules. Many vendors claim to have open architectures, but they are only open if one adheres to their proprietary standards. An architecture or framework can really only be called open if it is freely available and the command interfaces and data interfaces are widely published and easily accessible. The World Wide Web fits this picture.

Another major barrier to the imperative is the multiplicity of standards for specific domains and the length of time it takes to create a new standard. While the availability of certain standards will enable the move to information infrastructures, there are many conflicts and many standards are lacking. There are multiple overlapping standards in high-level communications, particularly for passing of messages among distributed objects on heterogeneous systems and for data exchange (Table 2). There are also diverse standards for graphical user interfaces for documents, images, video and sound. These are all important for next-generation manufacturers. The multiplicity of standards makes access and use of the data difficult. This particularly true in the design world where AEDES and ARGOS needs to be implemented. STEP has come a long way in addressing these problems but it is taking too long and has become so complex that users are adapting other standards to allow them to progress.

Table 2: Current and emerging multiple standards that form a barrier to responsive NGM information systems (NGM 1997)

Topic	**Subtopic**	**Example available or emerging standard**
Communication		CORBA, http, OLE/COM/DCOM, DCE, ISO/OSI
Data exchange	Product	STEP and its various APs
	Geometry	STEP AP 210, DXF, STL, HPGL, ACIS SAT
	Text, hypertext	http, SGML, XML, RTF
	Images	JPEG, BMP, GIF, TIFF, DIB, PCX, MSP
	Simulation data	SAVE MDF/ CDF
	Production plans	ALPS, STEP AP
Presentation/ GUI		Windows, X-Windows, Open GL, Tcl/Tk
Process modelling		IDEF, NIAM
Computer languages		C, C++, ADA, Java, Visual Basic
Sound		WAV, AU, RA, MIDI, RMI, AIF
Video		MPEG, AVI, MOV
Database Query Languages		SQL, SDAI

If there were standards for every aspect of the command and data interfaces among modules, this would not be a barrier. Lacking these, it currently takes too long for a group of companies to agree on even the product data exchange standards let alone all the other standards that must be considered. For example the Distributed Computing Environment Group (DCE), the Object Management Group (OMG) and Microsoft are all working on protocols for message passing between distributed objects on heterogeneous systems. The HTTP Working Group is working on similar standards. The contributors to the NGM (Agility Forum 1997) project are of the opinion that CORBA, OLE and DCE will merge with http and Java having a strong influence on all. Java and http will have a strong influence on the whole communication domain, but it is not yet clear what it will be.

During a recent investigation by the author of 16 companies that positioned themselves in the KM domain it became clear that the IT industry has already done a tremendous amount to gain a deeper understanding of the exact requirements and architecture of a Knowledge Management System (KMS). The solutions investigated ranged from attempts to integrate the multimedia enterprise knowledge in various forms in a readily accessible user interface to intelligent cross-linked repositories with highly configurable knowledge profiling environments.

All the local suppliers are unanimous in their opinion that it is at this stage almost impossible to fully quantify knowledge that originates at the tacit level. Tacit knowledge is seen as the experiential knowledge of people that exists mostly in the minds of professionals. For this reason all the solution providers are concentrating on the communications and access profile aspects of KM.

The main technical requirements to implement a KM system successfully can be summarised in Table 3. In order to implement a comprehensive briefing and design system the infrastructure needs are very similar to the requirements for KM. The only difference that can be identified is the type of knowledge that is stored and the mix of software tools that would be used to solve the various problems in the design stages.

Table 3: The main requirements for a Knowledge Management enabling environment (Collated by author)

Main Requirements	**Expanded software requirements for integrated knowledge based architectural briefing and design system**
Communication	**Network infrastructure** • Flexible high speed network configuration supporting distributed objects and an ubiquitous service environment. **General office knowledge content management** • Store all forms of project and office knowledge such as scanned paper documents, electronic word processing documents, video and voice recordings into an information base. • Provide support listing, browsing, sorting, grouping, filtering and searching of the knowledge base. **Teams that collaborate in real time and over distance** • Conversation services with transcript functionality for distance discussions. • Video conferencing for virtual meetings. • Screen sharing services for sharing of the document creation process, virtual white boards and application sharing. • Streaming media services for recording virtual meetings and video (meeting) on demand. • Event and meeting databases for organising and optimising meetings. • Home pages on Web servers for each task community, team or expert to speed up the access to project information (knowledge sources)
Design team flexibility and responsiveness	**Team skills profiles** • Directory and membership services that support the building of communities through grouping people together into expert teams working on the same set of information. • Forum services to create workspaces for communities and teams that contain all interest-related data. • Self-subscription services to specific matters of interest for dependent information delivery and subscribing. • Organisation databases integration like people skills and human resource databases for enhancing community, team and experts information and searches this information. **Responsibility and accountability** • Services to assign responsibilities to members of the team.

	• Workflow services for automatic trace ability processes based on roles and Subject Matter Experts (SME). • Tracking services that follow team contacts and team activities. • E-mail services for automating notification, routing and simple workflow services.
User Interface and information search	**User interface** • Personalization systems that allow customisation of the computer project interface. • Web browsers with the ability to include e-mail, project data and design intelligence tools for easy access. **Ubiquitous demand driven design and construction information** • Construction industry catalogue and search services. • Availability of material and product databases. • Acquire pre-packaged starter kits from other companies via the web. • Services to build own office electronic storage that combines often used information from external origin. • Notification services that react on changes in design or fundamental information contained in local catalogues and integrate with the e-mail system. **Reporting** • Dynamic project reports available on the Internet or intranet. (OLAP technologies and services) • Configurable ad-hoc, multi-media query builders that can be profiled to serve the needs of the user or the task at hand.
Project resource integration and access	**Information access** • One convenient entry/ access point to all project information and applications. • Knowledge indexing services that can index all the documents for easy subsequent retrieval. • Subscriber services where the service provider maintains the infrastructure and the subscriber accesses the service by means of his Internet Explorer/ Netscape driven by credit card payments, if and when required. This saves the user buying less often used specialised tools to analyse energy and cost if and when required. • Rapid design concept selection for new designs and projects based on previously stored structured knowledge. **Life cycle continuity of information** • Building design and operational information that is maintained over the life cycle of the building. • Captures project briefing and design decisions and knowledge automatically and during the course of the project in the form of plug and play design starter kits.

3.1.4.1 Hypertext based systems

One of the most basic means to organise electronic knowledge is by means of hypertext driven systems. An example of this is the use of a World-wide Web page with hyperlinks that could point to documents such as word-processing, spreadsheets, images and presentation documents. The capabilities of the Microsoft Windows operating system facilitate in-place activation.

The main advantage of hypertext is that the reader can choose his own associative way through hypertext, depending on his background knowledge, interests, context of use and his task at hand. One of the difficulties for the reader of hypertext is to get an overview of the structure of the knowledge presented. To this end hypertext systems normally provide tours.

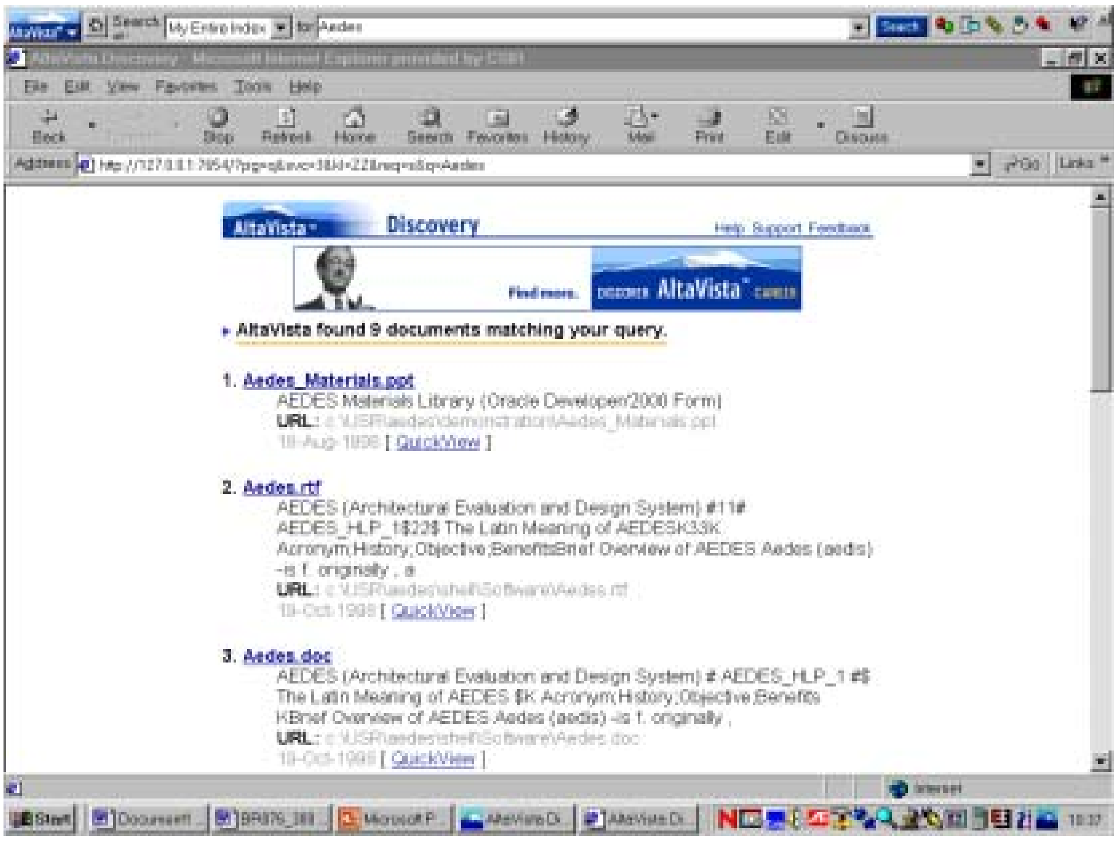
History lists provide intelligent backtracking. Bookmarks also assist to orientate the user in the vast hypertext landscape.

In *AEDES* a context sensitive help file was created that tested some of the underlying principles mentioned above. The present help file was created in the Microsoft .HLP format. This format will very likely be superseded with a HTML, www format help file. At the moment the .HLP format is predominant on the desktop, but the gradual move to HTML format is becoming evident in the Microsoft Office 2000 suite.

3.1.4.2 Search engines such as Alta Vista Discovery

The downloadable Alta Vista Discovery search engine provides a convenient means to index documents on the personal computer and in a network environment. The author tested the software and found it a good solution. The only problem is that the index needs to be rebuild from time to time to keep it current. This action takes a long time, typically an hour on the present equipment that we are operating on. The look and feel of the output is exactly the same as the Alta Vista search engine used in the Internet environment (Figure 6). In the example the search word was "Aedes". The response was a list of all documents that contain the word "Aedes". In the example Powerpoint, rich text file and Microsoft Word formats were found along with the local directory paths. The search engine is also capable to index email messages. A very rudimentary summarise facility is provided, but unfortunately not of much use, because it purely extracts random fragments of text up to the specified number of words.

The main advantage of this environment is that it can exist independently of any other system software. It provides good search and find facilities for a diverse range of documents. It can almost be seen as a very basic document management system.



AltaVista found 9 documents matching your query.

1. Aedes_Materials.ppt
   AEDES Materials Library (Oracle Developer/2000 Form)
   URL: [illegible]
   18-Aug-1998 [QuickView]

2. Aedes.rtf
   AEDES (Architectural Evaluation and Design System) #11#
   AEDES_HLP_1$22$ The Latin Meaning of AEDESK33K
   Acronym;History;Objective;BenefitsBrief Overview of AEDES Aedes (aedis)
   -is f. originally , a
   URL: [illegible]
   19-Oct-1998 [QuickView]

3. Aedes.doc
   AEDES (Architectural Evaluation and Design System) # AEDES_HLP_1 #$
   The Latin Meaning of AEDES $K Acronym;History;Objective;Benefits
   KBrief Overview of AEDES Aedes (aedis) -is f. originally ,
   URL: [illegible]
   19-Oct-1998 [QuickView]

Figure 6: A typical Alta Vista Discovery screen (Author)

3.1.4.3 Essential elements of a knowledge management architecture

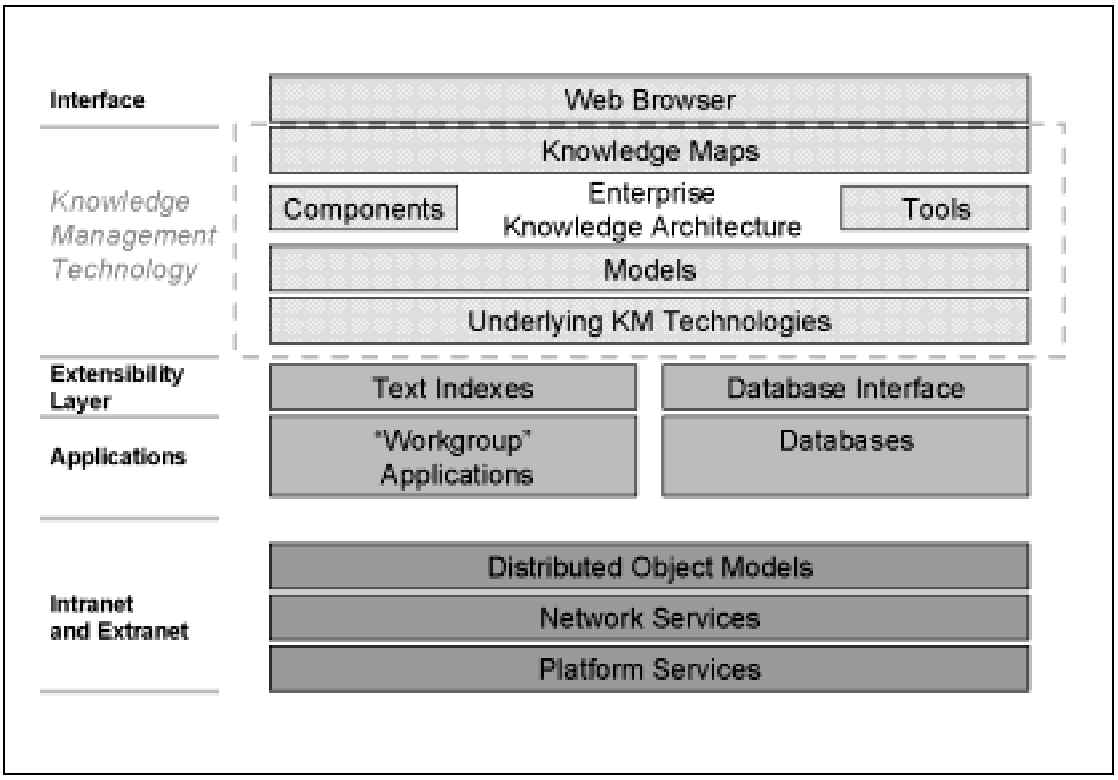


Figure 7: Knowledge management architecture (GartnerGroup 1998:3)

Figure 7 details the technologies that are targeted at KM. They are classified according to the architectural fit and the KM process that they support. The enterprise infrastructure is a networked service supporting distributed object models such as the OMG CORBA and Microsoft DCOM. Directory services and security services are becoming openly accessible by the upper levels of software. Workgroup applications continue to be silos of unstructured, mostly textual information that is set apart from database silos of structured information. Progress is being made to bridge the text data wall through SQL. However currently KM must be satisfied with the integrated retrieval of records. Text indices are easily offered using the centralised model of the Internet, but databases demand complex program interfaces and additional structures such as data warehouses.

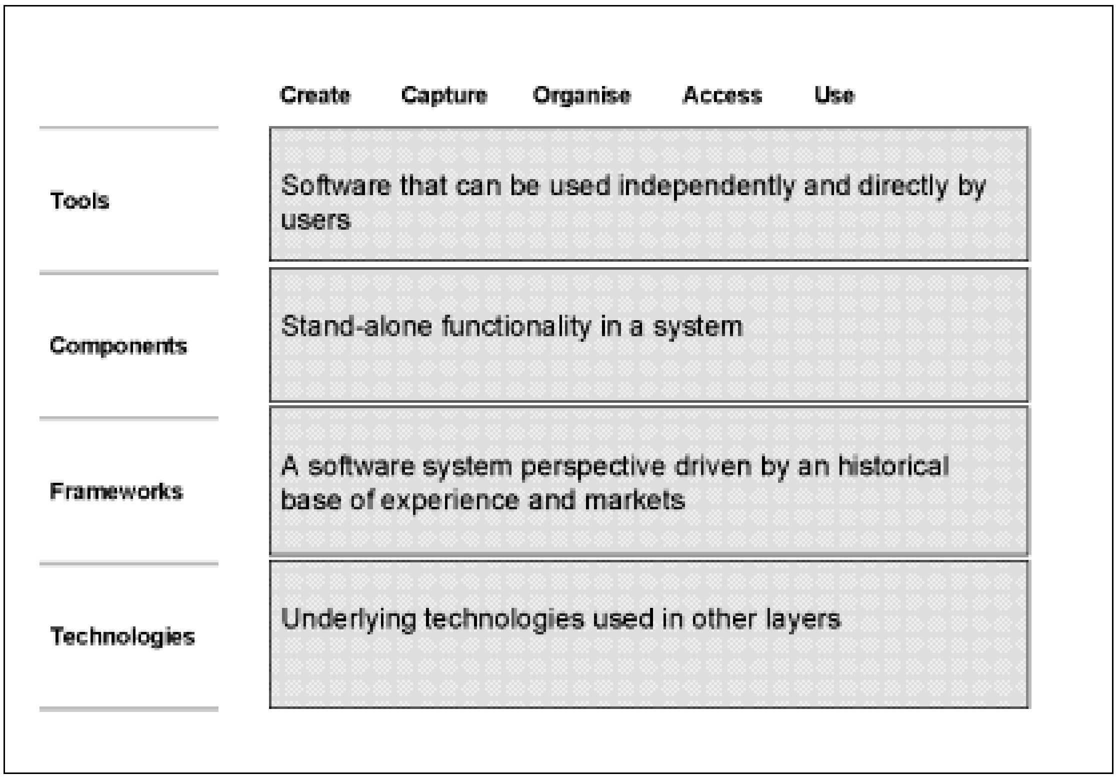

Create
Capture
Organise
Access
Use
Tools
Software that can be used independently and directly by users
Components
Stand-alone functionality in a system
Frameworks
A software system perspective driven by an historical base of experience and markets
Technologies
Underlying technologies used in other layers

Figure 8: Knowledge Management technology model (GartnerGroup 1998:4)

The KM technology layer of the architecture defines four essential elements (Figure 8). *Technologies* include mathematical algorithms, statistical techniques and database architectures. *Frameworks* define the central focus of an architecture, reflecting many subtle differences in vendor orientation. Information retrieval vendors are likely to be document-centric and more experienced with unstructured content. Database vendors are likely to be data-centric and experienced in high-speed transaction processing. *Components* are units of functionality that can be plugged into other products, but are not directly used by the knowledge worker. Tools are directly manipulated by the end user and are often branded. Good examples of desktop productivity tools are Visio, Word and Powerpoint. The software available today provides different levels of support for each of the five process steps. However *neural networks, Baysian Nets* and *linguistics-natural language processing* are all unproven. *Neural networks*, a development of artificial intelligence, learns to recognise patterns. *Baysian Nets* is a probabilistic model from past to future events. *Linguistics-natural language processing* is the representation of everyday language for computer understanding.

New deployment of advanced, underlying technologies presents a conundrum. The technologies are unproven but demanded by the market. *Natural language processing* (NLP) development and *neural network* application are both being funded by the same agencies such as the U.S. Department of Defence's DARPA (High Performance Knowledge Base Program. Neural nets require training that makes it very difficult to succeed with single pass queries. NLP is still the most promising for capturing the meaning of language, a capability that will soon be essential for KM.

3.1.5 Microsoft's approach to KM

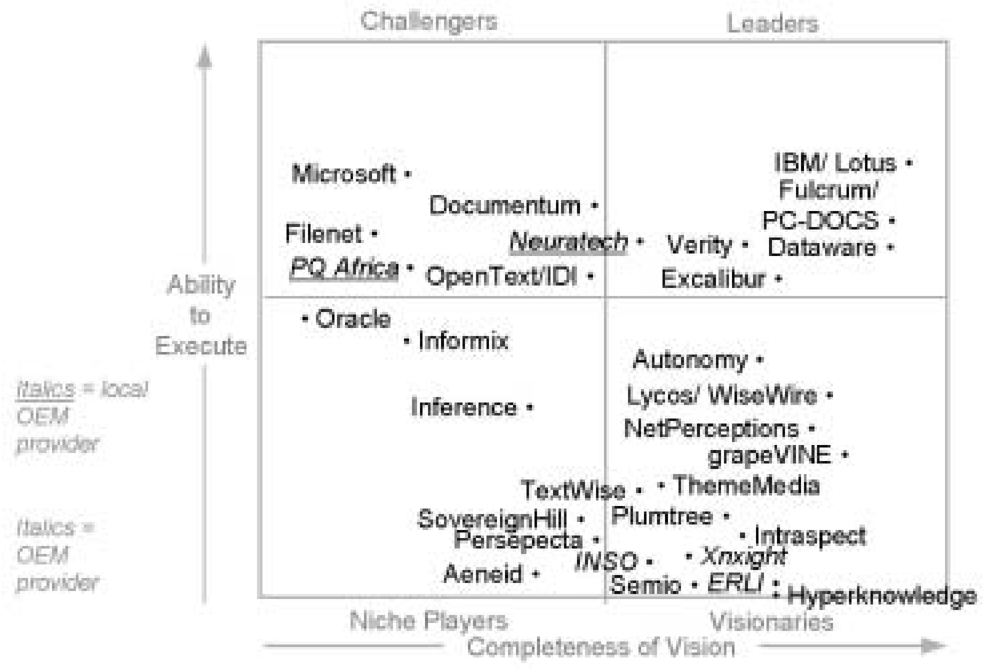


Figure 9: Knowledge Management vendors (Based on GartnerGroup 1998:16)

Although Microsoft remains strongly tied to the desktop, recent innovations in enterprise OS and network technology indicate that they are in the process of functional leapfrog. Figure 9 indicates the current position of Microsoft in comparison to other software developers with regard KM software. However in the present study the slight lack of vision as indicated is more than adequately compensated by the excellent desktop, document centric, convenient and portable object technology. Microsoft currently offers a comprehensive platform, albeit slightly conservative in terms of what a fully functional KM system will require. To ensure the success of the KMS very special professionals will be required such as Knowledge Architect (KA) and chief Knowledge Officer (CKO).

The Microsoft approach to KM is not unlike the generic structures illustrated in Figure 7. They are also of the opinion that the two prerequisite technologies for all KM systems are a *Complete Intranet* and *Messaging and Collaboration* (Leibmann 1999).

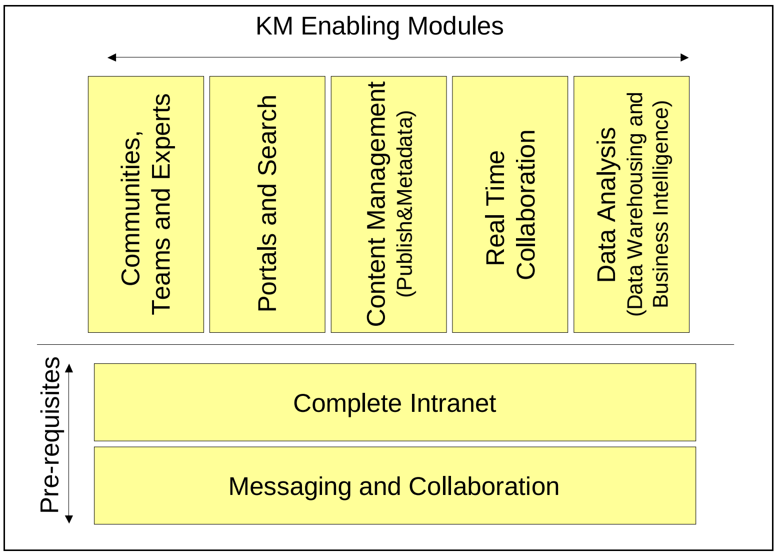

KM Enabling Modules
Communities, Teams and Experts
Portals and Search
Content Management (Publish&Metadata)
Real Time Collaboration
Data Analysis (Data Warehousing and Business Intelligence)
Pre-requisites
Complete Intranet
Messaging and Collaboration

Figure 10: The modules of a Knowledge Management evolution (Leibmann 1999:7)

The remaining KM-Enabling Modules extend the basic infrastructure to a KM system that includes services like *Content Management*, *Information Delivery* and *Data Analysis*. Automated services such as *Data Tracking* and *Workflow* processes are also included as part of the *Community and Team* competencies.

The KM-Enabling Modules have a modular character. Although some of the modules profit from the implementation of a previous module, they can be chosen in any order related to the specific business case that needs to be accomplished. Real-time Collaboration services, such as video conferencing, can be included on top of the pre-requisite technologies, but are enhanced by the meta data services provided in the *Content Management Module*. Meta data is a special database of where diverse sources of data can be found.

3.1.5.1 Messaging and collaboration

IT systems intended to support KM need to support the capturing of undocumented information such as human thoughts, sharing of ideas and documents. It is important to find this information efficiently. Another prerequisite of an IT system that supports KM is the existence of a set of common tools that are well known by all knowledge workers.

The tool that is used to provide an entry point to this IT system presents the information and controls all interaction with it. It needs to be capable of handling all the information that is part of the working environment of the knowledge worker. Ideally only one tool or application should exist for this interface.

The entry point to the information and applications in a KM system is also called a “Portal Service”. If the same environment also supports also the creation of content, it is called a KM desktop. The capabilities of web browsers make them ideal candidates for this task.

3.1.5.2 Complete Intranet

An information network that provides access to all the data needed supports this module of KM. Decisions must be taken fast enough to get a competitive advantage. A Complete Intranet KM system should enable people to find the right information or sources for helping solve problems or drive decisions.

3.1.5.3 Communities, teams and experts

The two pre-requisite technologies of KM put all collaboration and document-based knowledge sources together enabling the knowledge worker to browse information objects based on knowledge groups. *Communities, teams and experts* add the next level of sharing knowledge and turning it to results.

Teams differ from communities in that teams are task driven and communities are interest driven. A team usually works closely together, in a workgroup, on the same tasks and goals. In many cases the information produced by a team is closely held within the team until it has reached a level of completeness where it can be shared, for example in a review, with a broader audience. Communities are mostly driven by interests in the same area and are more loosely coupled, for example by subscriptions. Communities are especially useful for building knowledge to higher levels, often by getting successive levels of input from a wide audience.

The role of an expert is to qualify and filter information. Often an expert is related to a limited set of subjects. Subject matter experts (SME) can be defined in two ways. He is either an organisational function (defined by the KA) or as very knowledgeable person who is a well-known expert in his team or organisation, assuming the status of an SME for contributing high quality information or for reviewing it.

The SME is an important role for a KM-related Information Web or Intranet. In traditional Intranet solutions, there is little control over who can store or upload information into the Intranet. This is not a bad thing and is desirable in order to build an extensive information repository. To maximise the usefulness of the Intranet, the information should be filtered, classified and grouped. This process is part of the responsibility of the SME.

Communities, teams, and experts are also used for the controlled process of putting information into the KM system. Filtering, qualification, approval or more complex workflow processes for documents and other electronic data need to be established. In a KM system these processes are not strictly based on traditional organisational roles such as manager, reviewer, approver and author but more on Subject Matter Experts. This can add a great level of dynamic and flexibility to the KM system and the automated processes.

3.1.5.4 Portals and search

Portal Services like Yahoo, Lycos and Excite are well known. They allow consumer oriented services such as easy information shopping. Those Portals categorise personal interests in groups like “News”, “Sports”, “Economy”, “Education”, “Science” and “Entertainment”. They allow for easy browsing within those groups in building a logical hierarchy of subgroups or forums. The browsing and search services support the consumer in his quest to gain knowledge out of the large Internet information store. Another benefit of these consumer Portals is the high customisation provided for its visitors. Objects of interest can be bookmarked in a personalised Portal, allowing for immediately access when revisiting the portal site.

This technique when applied to business-oriented goals is one of the key KM-enabling modules following the same idea of the consumer-oriented portals in the corporate world. Business Portals provide the knowledge workers within the company, and also external suppliers and customers, with instantly task-relevant information objects. A Primary goal of a portal is the transparent enterprise, hiding the complexity to access knowledge stores. Even if the user accesses legacy information stores he should not be aware of it.

Examples of Business Portal Information Objects are:

- Corporate and team links.
- Team application links.
- Incoming mail notification and headers.
- Personal tasks.
- Corporate search.
- Integration of business intelligence data.

From the examples, some direct organisational tasks can be derived. Teams in the enterprise need these definitions in order to locate internal or external company information. This allows them to successfully include links to that information into the portal.

This Module also defines the creation of catalogues that build groups of related information based on business needs over structured and unstructured enterprise information (KM information base) to allow for full-text search against the partitioned data. An extension to the Catalogues is the definition of Searches against these Catalogues.

In order to define the Catalogues for an organisation, there has to be a very good understanding of the business and its processes. At this stage, the Knowledge Architect needs the support from the different divisions, business units and departments that understand how their information is organised and is related to their business goals, tasks, and needs.

3.1.5.5 Content management

Portals and Search address the problem of searching knowledge using all information sources in the enterprise such as structured and unstructured internal information objects. Examples of these are office documents and collaborative data. External sources such as partners, suppliers and competitors can be identified. Other external sources such as the Internet provide a tremendous potential for knowledge if the criteria for including such information are well chosen.

All the pools of information sources that are part of and accessible to the KM system combine to build the KM information base. This Module handles how knowledge assets get into the KM information base. To handle this new complexity of the KM information base and help knowledge workers to stay focused on solving business problems without disappearing in technology a sophisticated KM taxonomy needs to be built based on meta data. It also needs to publish information in the knowledge base. The KM information base must then be made accessible through operations driven by the meta data complex.

When publishing, several things should be considered concerning the KM taxonomy. Meta data tagging of documents is important for the quality of the content in the stage of document publishing. However it should not be a burden for people to submit information. The KM system must encourage users to submit information. Positive aspects for promoting this condition are the building of well-focused Communities in order that users feel part of and respected in a concentrated team and do not loose their inclination or motivation to submit. Building huge submission and posting systems where users do not get recognised or rewarded will discourage them from providing their knowledge, which will prevent the company from evolving a culture for knowledge management.

3.1.5.6 Real-time collaboration

The knowledge on a specific subject is often not in documented form and is therefore lost to the organisation. There are ways retrieve the lost knowledge into a state where an IT system can manage it. This especially focuses on areas where computers can be used to exchange thoughts, documents and other aids for capturing such tacit knowledge for the KM information base.

The process of capturing tacit knowledge can start with the introduction of simple computer-based chat services. Regular meetings arranged with expert groups to talk about specific topics can be extended with these services, well known from the Internet and enriched by building automatic transcripts for the chat sessions. Transcripts can be enriched with the corporate KM meta data and stored in the KM information base for later search and retrieval.

More advanced services like video conferencing follow the same concept. The video stream is recorded on video equipment and is subsequently transferred to the KM system. Descriptions and meta data are either merged with this video stream or can be stored in parallel on a file or database. For cultures where such virtual meetings are common, an event database is typically built where upcoming and past meetings are stored, together with event titles and descriptions. They can be listed or searched by subject matter by means of meta data. A hyperlink is provided so those users may join a virtual meeting. If the meeting takes place in the future, integration into the e-mail system ensures that this event is marked in the calendar, and on the event date a reminder automatically guides the participant to the virtual meeting. After the event or meeting, on-demand services will make that knowledge available, by providing the recorded video out of the KM information base to the KM desktop.

When integrating this technology into the automated KM services scenario, notifications are sent automatically to the appropriate knowledge workers to remind them of an interesting meeting or event. The appropriate URLs can also be listed on the KM portal.

A hybrid of abovementioned technologies is the integration of presentation techniques. In this an online presentation that consists of slides is sent over the network. The audience receives the video, audio and slides of the presentation on the KM desktop. The chat service is integrated as a separate area on the KM desktop and enables the audience to type questions during the meeting into the chat area. These questions are transferred to the presenter or a person controlling the online presentation. On receiving the questions the presenter can answer them during or at the end of the event. The slides as a document, the chats as a transcript document and the audio and video as a stream are linked together and stored in the KM system.

The same technologies not only make virtual events available for the KM information base, but also real events like conferences. Each session on a conference can be recorded and than be made available for all employees in the events system on the corporate net. Another solution is to produce CDs of the sessions and distribute them to all subsidiaries or make them orderable for interested employees.

Real-time Collaboration KM also provides support for sharing the creation process. It enables distant knowledge workers to share a single virtual working space and to collaborate on the creation of documents. This includes not only the sharing of the creation process using the productivity suite, but also white board functionality. This kind of technology is also known as screen sharing.

3.2 Knowledge based design

3.2.1 Introduction

This section investigates the various Knowledge-Based Design approaches in an attempt to see whether they could add value to the final solution proposed. Over the years many different approaches were used that had different levels of success. It is intuitively sensed that highly structural and prescriptive methods are not suitable. It also appears that certain information in design is well defined whereas other information is incompletely specified and vague. This section analyses Artificial intelligence and design, problem-solving architectures and Case-based design.

The first generation of Knowledge-based Design Systems (KBDS) was characterised by the dominance of logic models and rule-based systems then prevailing within expert systems technology. The paradigm of Knowledge Engineering (KE) appeared to be promising and relevant to design. KE turned out to be far more applicable to Knowledge Management (KM) that is likely to form the holistic operational framework for globally enabled design and project environments. KE has limited use for the range and complexity of design tasks. Debenham (1998:1) states that a unified KE methodology treats data, information and knowledge in a homogeneous manner. However, with a few exceptions, models of expert knowledge appeared to have limited utility for the range and complexity of design tasks (Oxman *et al.* 1994).

Debenham (1998:23) defines a Knowledge-based system as a system that represents an application containing a significant amount of real knowledge and has been designed, implemented and possibly maintained with due regard for the structure of the data, information and knowledge. A significant amount means that the application boundary of the system should identify an area of the application that is appropriately dealt with using knowledge-based systems design techniques.

In an application:

- Data is the set of fundamental, indivisible things (Debenham 1998:18).
- Information is the set of implicit associations between data things (Debenham 1998:20).
- Knowledge is the set of explicit associations between the information things and/or the data things (Debenham 1998:20).

An expert system is a system in which knowledge is represented as it is, possibly in the same form that it was extracted from an expert. In an expert system the represented knowledge should endeavour to solve problems in the same way as the expert knowledge source solved them.

Debenham (1998:23) identifies differences between Knowledge-based systems and expert systems:

- Expert systems perform in the manner of a particular trained expert. A knowledge-based system is not constrained in this way. In a knowledge-based system the represented knowledge should be “modular” in the sense that it can easily be placed alongside knowledge extracted from another source.
- Expert systems do not necessarily interact with corporate databases. In general, knowledge-based systems belong on the corporate system platform and should be integrated with all principal, corporate resources.

Carrara *et al.* (1994) states that computer-aided architectural design research has inherited the unanswered questions first raised by theorists like Rittel, Simon and Schon. To this has been added the additional complexity of representing the answers in an explicit and complete way so that they can be handed over to and reproduced by machines. The combined search for solutions to these questions is called Knowledge Based Computer-aided Architectural Design (KBCAAD). This title implies that the search for tools that could assist designers in the design of buildings relies on one hand on understanding the cognitive processes of architectural design itself as well as the theories, methods and techniques that have been developed outside the discipline of architecture. It is the synthesis of these two sources that holds the promise that an appropriate balance will finally be found.

Architectural CAD researchers have been focussing their attention on the cognitive aspects of the architectural design process since approximately 1990. They have been constructing models of design knowledge and reasoning. They developed data structures to represent them computationally. Although the models are unique to the discipline of architecture they were borrowed and adapted from other disciplines such as Artificial Intelligence (AI) and product development. Inference engines that were prominent at the height of the interest in expert systems have not proved themselves for design applications of substance.

Due to the complexity of design, systems for design have often defined the task with artificial narrowness (Hinrichs 1991:3). In AI, as in Fuzzy Set theory, progress in the past was made by limiting the universe of discourse or even closing it in an attempt to simplify the enormously complex design problems. To make the systems tractable the following typical four approaches were used (Hinrichs 1991:3):

- *Selection.* Select components to instantiate a skeletal design. Selection problems are typically constraint satisfaction problems in which all variables to be satisfied for are known ahead of time. The space of possible components is given as if from a catalogue.
- *Configuration.* Arrange a given set of components. Configuration is essentially the dual of selection. This method concentrates on the relationship between components rather than the components themselves.
- *Parametric.* Fix numeric parameters. Parametric problems are similar to selection problems except that the components are quantities and the task is usually to optimise or to partially satisfy constraints.
- *Constructive.* Build up designs from components. Constructive design is analogous to planning, in that components take on the role of operators or actions. Typically, the space of components is fixed throughout the design process.

Hinrichs (1991:3) observes the fact that if design problems are viewed as instances of abovementioned types, they can often be solved using efficient algorithms and heuristics. However, rigid classifications do not capture the flexibility that real designers exhibit. A model of design or a system that supports design in an open world should be able to use any of the four generic types of design.

In addition to the different types of design approaches, AI research has explored different approaches to the process of design. Hinrichs (1991:3) summarises some of these approaches as:

- *Pure synthesis*: Construct designs from the bottom up. The pure synthesis approach assumes that the design problem space is basically a very large graph. If the appropriate heuristics can be found to prune it, searching that graph can discover solutions. An example of this type of approach can be found in the rule and production based systems such as LOOS (Flemming 1994:5). It contains a generator able to accept a layout and find all possible ways of adding a new object. A tester evaluates a layout generated in this way and a controller mediates between these two components. After each generate-and-test-cycle, the designer selects the next layout to be expanded based on the evaluations produced by the tester.

- *Hierarchical refinement*: Refine skeletal designs from the top down. Hierarchical refinement assumes that there are really only a few basic types of designs. If the problem can be classified, a design template can be instantiated, and propagating constraints can solve variables. An example of this is the simple cut-and-paste approach developed by the Division of Building Technology of the CSIR in South Africa for health facility design. It is based on a large set of templates at various levels and assists unskilled designers to rapidly design a hospital or a clinic.

- *Transformational approach*: In this approach design is claimed to be a mapping from function to structure. Just as Fourier and Laplace transforms map from one domain that is difficult to reason in to another that is more tractable, the transformational approach to design suggests decomposing functions and mapping primitive functions onto structures. Conradie and Küsel (1999) experimented with this in the precedent system AEDES discussed in Chapter 4.

- *Case-Based Design*: The case-based and analogical approaches assume that the problem being solved is probably similar to one that was seen before. If a historical case can be retrieved, a solution can be found by transferring directly from that previous case. An example is the ARCHIE-2 system (Goel *et al*. 1991; Kolodner 1993) that uses similar solutions from the past to solve the current problem.

Currently the most promising AI solution is the use of design cases. This is empirically validated successful solutions and failures to design problems from the past. If structured design methodologies are to be used then design knowledge generated should be stored in such as way as to expedite future designs. This is also one of the key objectives of Knowledge-Intensive CAD where attempts are being made to elevate CAD systems beyond only electronic drawing boards. Mäntylä (1995: 3) states that a key objective for the logistical management of information is reuse of existing information. Ideally all (design) information created or learned should be made available to later use in a correct, useful and timely fashion.

3.2.2 Artificial intelligence and design

In the late 1950s Allen Newell and Herbert Simon proved that computers could do more than calculate. Marvin Minsky, head of the Massachusetts Institute of Technology (MIT) Artificial Intelligence (AI) project at the time, announced with confidence that within a generation the problem of creating Artificial Intelligence would be substantially solved. Then suddenly the field of AI ran into unexpected difficulties. The trouble started with a failure of attempts to program an understanding of children’s stories. The program lacked the common understanding sense of a four year old and no one knew how to give the program the background knowledge necessary for understanding even the simplest stories. An old rationalist dream was at the heart of the problem. AI is based on the Cartesian idea that all understanding consists in forming and using appropriate symbolic representations. For Descartes, these representations were complex descriptions built up out of primitive ideas or elements.

Dreyfus (1993:*xi*) states *"Common-sense understanding had to be represented as a huge data structure comprised of facts plus rules for relating and applying those facts."*

AI struggles with essentially three central problems (Dreyfus 1993:*xviii*).

- How everyday knowledge must be organised so that inferences can be made.
- How skills or know-how can be represented as knowing-that.
- How relevant knowledge can be brought to bear in particular situations.

Dreyfus (1993:*xxviii*) states that *"Heidegger, Merleau-Ponty, and the gestaltists would say that objects appear to an involved participant not in isolation and with context-free properties but as things that solicit responses by their significance."*

"What we really need is a system that learns on its own how to cope with the environment and modifies its own responses as the environment changes. To satisfy this need, recent research has turned to an approach sometimes called 'reinforcement learning'." (Dreyfus 1993:*xxxix*)

"The point is that a manager's expertise, and expertise in general, consists in being able to respond to the relevant facts. A computer can help by supplying more facts that the manager could possibly remember, but only experience enables the manager to see the current state of affairs as a specific situation and so see what is relevant. That expert know-how cannot be put into the computer by adding more facts, since the issue is which is the current correct perspective from which to determine which facts are relevant." (Dreyfus 1993:*xlii*)

Feigenbaum makes the following comments in his analysis of MYCIN, a program developed by Shortliffe in 1976 for diagnosing blood and meningitis infections and recommending drug treatment (Dreyfus 1993:28).

" He conscientiously notes that the experts themselves are not aware of using rules:

...Experience has also taught us that much of this knowledge is private to the expert, not because he is unwilling to share publicly how he performs, but because he is unable. He knows more than he is aware of knowing. (Why else is the Ph.D. or the Internship a guild-like apprenticeship to a presumed 'master of the craft'? What the masters really know is not written in the textbooks of the masters.) "

The author tested the translation capabilities of a translation program freely available on the Internet. The result is really amazing, however upon closer inspection certain inherent and fundamental problems become apparent.

The following slightly technical paragraph from a recent German conversation class was submitted to the translator.

" Die Stadt Frankfurt am Main

In der Stadt Frankfurt gibt es heute viele grosse Bürogebäude und in der Umgebung findet man viel Industrie. Die Farbwerke Höchst, wo Farben, Lacke und andere Chemikalien erzeugt werden, sind in der Nähe von Frankfurt zu Hause. Andere grosse industrielle Konzerne sind Siemens und Halske AG., Hartmann und Braun (Elektroinstrumente) und Mouson (Kosmetik). Auch die Glas- und Porzellanindustrie ist bedeutend.

Frankfurt ist auch ein kultureller Mittelpunkt und hat natürlich eine Universität und verschiedene Hochschulen. Das Städel ist die städtische Kunstgalerie und besitzt viele Kunstschätze. "

After a few seconds the translator responded back with the following partially correct answer.

" The city Frankfurt/Main

In the city Frankfurt gives it today many large office buildings and in the environment finds one much industry. The inking attachments Hoehst, where colours, lacquers and other chemicals are produced, are in the proximity from Frankfurt at home. Other large industrielle of companies are Siemens and Halske AG, hard man and brown (electrical instruments) and Mouson (Kosmetik). Also the glass and porzellanindustrie are important.

Frankfurt is also a cultural focal point and has naturally a university and different universities. The Staedel is the urban art gallery and possesses many art treasures. "

Upon analysis of the results it is apparent that the problems mentioned by Dreyfus are real. The wrong translation of the phrase *"gibt es heute"* indicates that the context or idiomatic expression was not understood. The most serious error was the company *"Hartmann und Braun"* that was translated as *"hard man and brown"*. This indicates that purely mechanistic parsing was used in this case without any higher level contextual comprehension. It is also apparent in the short paragraph that the translator had difficulty with certain technical terms such as *"porzellanindustrie"*. It is the author's experience, and Dreyfus confirms it, that AI is only successful in small well-defined domains. The above translation does not make sense on its own. If AI is to be successful at all it is mandatory that context be well understood. Nobody has yet succeeded in devising an algorithm that can accurately summarise written text or a book, because this process would require exceptional contextual understanding. This is indeed one of the most difficult problems known.

Dreyfus (1993:*xxx*) notes *"It seems highly likely that the rationalist dream of representationalist AI will be over by the end of the century"*.

Designers and CAD researchers became interested in AI for two reasons. The first is the influence of structured programming as propagated by Dahl, Dijkstra and Hoare (Flemming 1994:1). Computer programming is seen as a process of step-wise refinements where program specifications are developed through several levels of abstraction. The specification is complete at any level. Transitions from one level to the next consist of expanding the program by adding greater detail. The prescriptions of structured programming are impossible to follow in many design situations because they presuppose that the task at hand is well understood and amenable to algorithmic treatment.

The second reason was due to the frustrations with the unintelligent nature of commercial CAD systems. Even today CAD is contributing very little to the initial and most demanding stages of design. In an attempt to solve the latter AI was applied. AI is generally concerned with tasks whose execution appears to involve some intelligence if done by humans. Design falls into this category.

AI research can be divided into two broad approaches.

- *Understanding of the human brain.* Computer models in this tradition represent a model or simulate human cognition and succeed to the degree to which they emulate human performance.
- *Intelligent systems.* Systems that perform intelligent tasks effectively without concerns for how faithfully the model simulates human performance or cognition.

The efforts in the first category are theoretically motivated and must seek empirical acceptance. Efforts in the second category are practically motivated and must stand the test of practical usefulness. Computers that work exactly like people are unlikely to do better than people. CAD tools, whether AI based or not, should always be seen as a complement to human designers assisting them in tasks where they perform less well, but do not compete in areas that the human brain performs well. Programs that assist in design are most useful in the following areas:

- Suggest possibilities to designers they have not thought of.
- Remind them of things they might have forgotten.

The author will attempt to prove that in addition to the two possibilities a third option exists. This is where intelligent components are used to facilitate the manipulation of complex design information in a convenient environment to facilitate concept selection and design experimentation during the early phases of design. During this phase the designer is often confronted with incomplete information and designs could very easily change. At the same time decisions taken during this phase will significantly influence operational characteristics.

Flemming (1994:21) states that the fixation of some AI researchers on processes rather than results is puzzling. Chess-playing programs were initially considered a legitimate AI topic. Recently the world chess champion, Kasparov was beaten by the super-computer Deep Blue. This was not achieved by imitating chess players, but rather by a more efficient generate-and-test approach that results, in part, simply from hardware improvements. In addition there was behind-the-scenes expert intervention. Bellman (1978:144) came to the conclusion that the human brain remains far above anything that can be mechanised. Oksala (1994:27) states that many architectural problems are life-long and in a theoretical sense typically non-terminating. Architectural products are often so complex with respect to environmental situations that we can only describe them partially.

3.2.2.1 Life cycle enabled design ontology

In the design and implementation of software intended to support the design and construction environment, ontology plays an important role. Simoff *et al.* (1998:23) mention that ontology originated in philosophy as a systematic account on the nature and the organisation of reality. Currently ontology is considered to be a branch of metaphysics addressing issues such as the categorical structure of reality. There is no ontology that is accepted as the definitive categorical scheme. A typical categorical scheme has a hierarchical structure with the most general entity at the top of the hierarchy.

Simoff *et al.* (1998:23) mention that the concept of ontology entered the field of artificial intelligence as a formal system for representing domain concepts and their related linguistic realisations by means of basic elements.

Unfortunately there is growing confusion about the meaning of the term in the context of its usage in AI in design. Presently the term is so wide that it ranges from the ISO STEP object model description (a hierarchical interoperability standard) to concept structures for sharing ideas. The application of ontology for the description of design domain faces additional difficulties due to the interdisciplinary and evolutionary nature of the domain (Simoff *et al.* 1998:24).

In the precedent systems PREMIS and AEDES three fundamentally distinct ontologies can be identified.

- *Location (property)*. This is a hierarchical structure that expresses the hierarchical relationship of locational entities. It starts with the *country* definition and goes right down to *shared space* (Figure 11).
- *Administration*. This is related to the activities and administrative uses that are made of these locational entities for managerial, classification or maintenance purposes.

- *Graphical objects*. The graphical database contains a set of structured entities that links to the alphanumeric relational database.

Due to the fact that *location/ property* is pivotal to Facilities Management portion of the life cycle this world was viewed as primary in PREMIS. *Administration* is broken down into the main categories *organisational structure*, *legal*, *people* and *construction elements* (Figure 12). The relationship between *graphical objects* and the *location alphanumeric* category is maintained by means of a system of implicit linking (Appendix A). In this system a graphical object has a relationship with an alphanumeric record by virtue of the fact that the graphical object name is the same as the concatenation of the key fields in the relevant Relational Database Record.

In an attempt to structure briefing and design Conradie and Küsel (1999) used an oversimplified ontology in the AEDES prototype of *active* and *passive* requirements and functions. Passive requirements are related to physical elements such as *structure*, *services*, *finishes*, *fittings*, *furniture* and *equipment*. Active requirements are activities that were given activity function names such as *enable ablutions*. In this approach SE principles as well as functional decomposition were explored.

Simoff *et al.* (1998:28) suggest an ontology that delineates the categories of building design space as *activity* and *space*. An *activity* consists of *equipment*, *service*, *time*, *performer*, *consumer* and *constraints*. Space consists of *geometry*, *divider*, *link* and *constraints*. In this model relations define the explicit connection between entities.

In Product Data Modelling (PDM) that is essentially the briefing and design phase of the product life cycle another important world can be identified:

- *BPM structure*. This is a traceable hierarchical framework for organising design knowledge and documentation on every stage of the design process. A good example of this is the Industry Foundation Classes (IFC™ Release 2.0) as defined by the International Alliance of Interoperability (IAI). This is essentially a hierarchical object structure that describes structural interactions, but also attempts to facilitate interoperability between different systems in the construction industry. This model is not as comprehensive as the ISO STEP object standard (Figure 13).

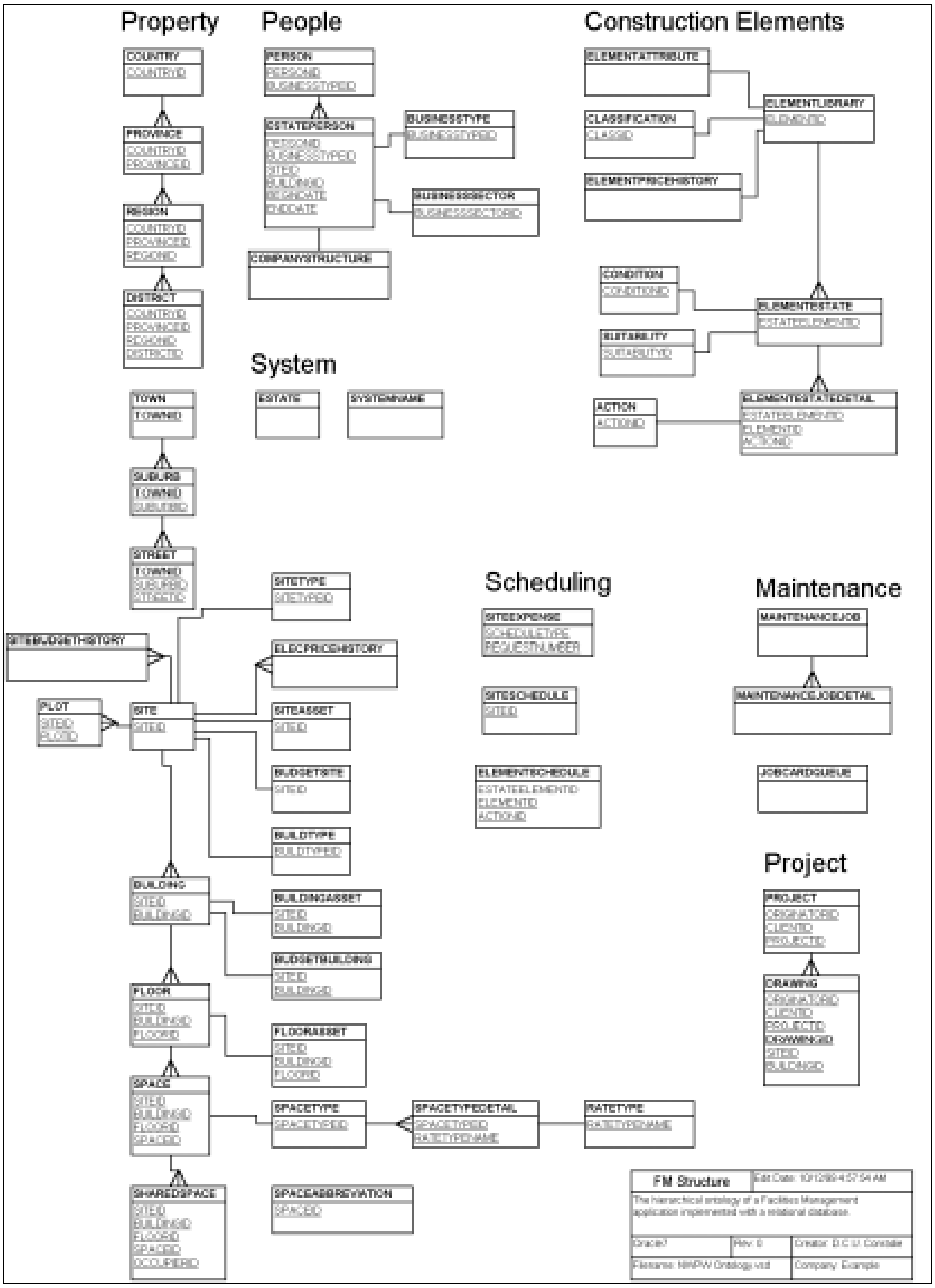


Figure 11: Typical hierarchical relational database structures used in a Facilities Management system (Author)

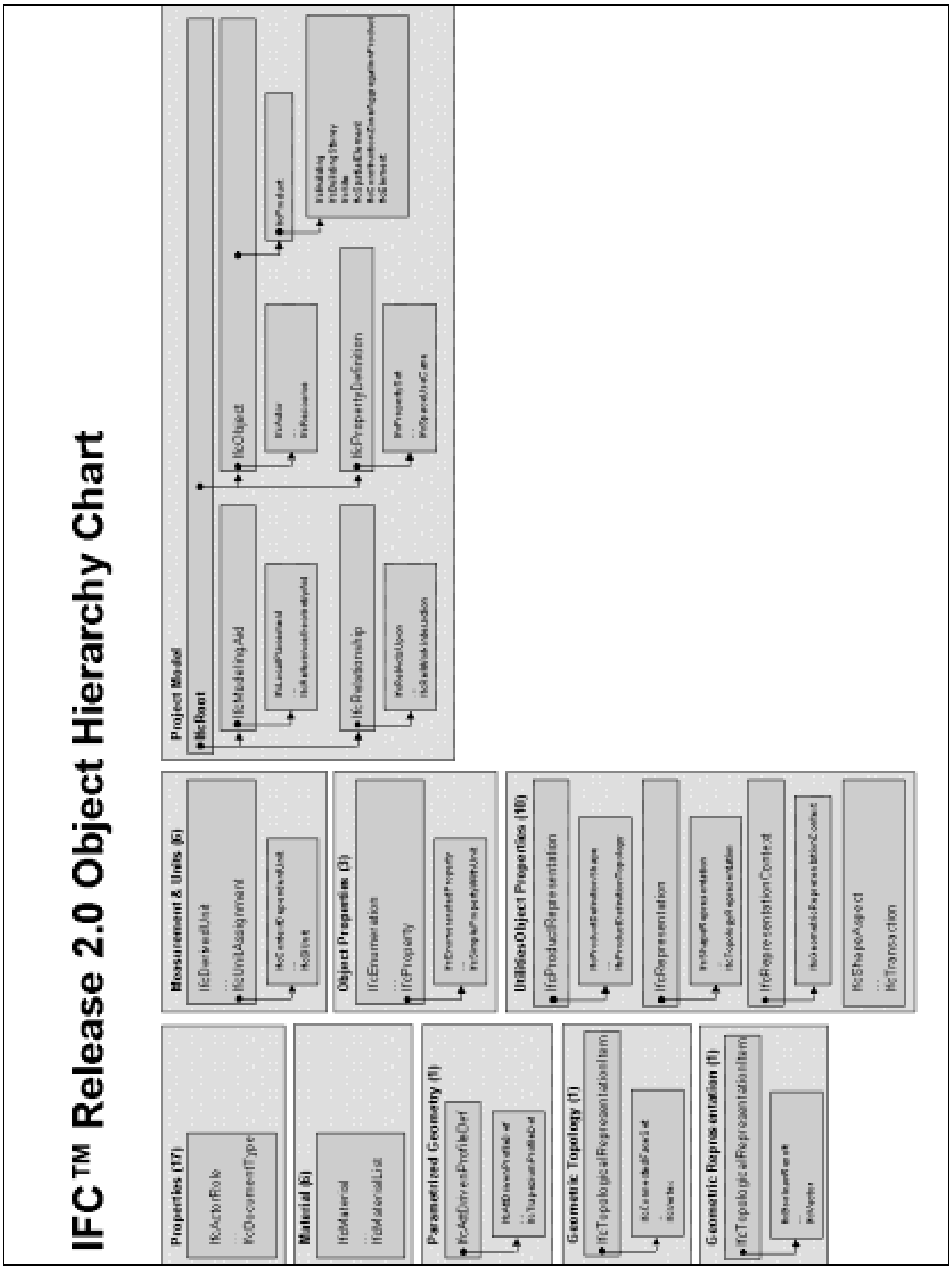


Figure 12: IAI, Industry Foundation Classes Release 2.0 Object Hierarchy (Author)

3.2.3 Problem-solving architectures

Today the following general design problem-solving strategies exist (Flemming 1994):

- Top-down strategies that develop a design specification through several levels of abstraction.

- Bottom-up strategies that construct a design incrementally in small steps more or less at the same, final level of abstraction.
- Middle-out strategies that start with a highly structured description and, transform it to satisfy given requirements.

Within each of these strategies a function or behaviour driven approach could be distinguished from a form driven approach. These distinctions correspond approximately to the goal- and data-driven approaches in AI.

3.2.3.1 Top-down strategies

1. Function-driven strategies

The logic design component selects functional components from a knowledge base that contains descriptions of individual components and templates that tell the system how to design with them. This synthesis proceeds through the levels of a functional hierarchy. At the highest level, it derives the overall system architecture in terms of functional subsystems and ensures correct interconnections between subsystems. The next level inherits the characteristics of the components determined at higher levels and can split single functional components into successor parts. When the lowest level is reached the individual subsystem components and their connections are known. The AEDES prototype system is an example of this type of functionally based system.

2. Form-driven strategies

This strategy is closely related to the type of structures that the IAI (Figure 12) proposes. It is a decomposition of the building structure into the various sub-structures and materials. The fundamental approach is to build a model of a building and to generate all documentation from the 3D model. The process to design a structure for a specific building starts from specifications of the overall building form and type. It successively selects component types and materials at each level in the hierarchy. The choices available at each step can be found by one of three methods:

- Selection from a pre-defined, finite set of alternatives.
- Synthesis by further decomposition.
- Computation based on rules or numerical calculations.

Constraints can be defined to avoid certain combinations of decisions. A design approach in which the parts for assembly are selected from a predefined set is also called *configuration design*.

3.2.3.2 Bottom-up strategies

1. Function-driven strategies.

This approach takes individual functional, behaviour or performance specifications and derives a description of a design incrementally by taking these specifications into account. Flemming (1994) states that this approach is rare because the interactions between performance indicators and design variables are so complex that it does not generally render this approach feasible. One particular system (WRIGHT) avoided these problems by using a system of disjunctive constraints that translate the desired behaviour characteristics into constraints in the design variables. This system determines all feasible ways of satisfying the constraints incrementally using constraint satisfaction techniques developed in AI.

2. Form-driven strategies

Form-driven, bottom-up strategies are employed by the classical incremental generate-and-test approaches that generate a design in small steps. The intermediate evaluations are used to direct the process into the most appropriate direction. Typically it would contain a generator that is able to accept a layout and find all possible ways of adding a new object. A tester evaluates a layout generated in this way and a controller mediates between these two components. After each generate-and-test-cycle the controller selects the next layout to be expanded, based on the evaluations produced by the tester. An early experimental system that used this approach is LOOS. It is unlikely that future Case-Based Design methodologies would use this type of approach, because it is so tedious. The computer attempts to execute tasks that a human designer can do just as well.

3.2.3.3 Middle-out strategies

1. Function-driven strategies.

In this approach a system starts with a highly structured description of the desired behaviour of a design. This description is transformed, at the same level of abstraction or granularity, into a physical description. An example of this is a system that accepts a graph-based description of an algorithm to be executed by a computer chip and transforms this description into a collection of hardware components and their connections.

2. Form-driven strategies

This strategy starts with a highly detailed and structured design description and refines or adapts it to the given context. Examples of this are the ARCHIE and ARCHIE-II systems described in detail by Kolodner (1993). ARCHIE is an interactive prototype *Case-Based Reasoning* (CBR) system for the design of buildings such as libraries and courthouses. It supports the construction and evaluation of solutions. Users specify their problem description and/ or solution description. The system retrieves and displays past designs and provides suggestions and warnings. In support of evaluation, the system computes potential outcomes and retrieves and displays past designs with similar outcomes. ARCHIE showed that design cases could be very large and need to be decomposed into smaller units. Libraries of design cases can be useful but may need to be supplemented with other types of design knowledge. Practical support systems need usable interfaces to allow easy access to relevant information. The most important lesson learnt is that the operation should be kept simple. ARCHIE is a useful precedent for the present study even though Kolodner (1993:162) described ARCHIE as a failure.

Architectural design systems based on CBR must solve two major system design and implementation problems:

- *Indexing*. An indexing system must be designed to facilitate retrieval of stored cases so that the most appropriate ones can be retrieved in the new design situation.
- *Adaptation*. They must support the refinement or adaptation of an existing case to the new situation.

3.2.4 Case-based design

3.2.4.1 Introduction

A solution stored for possible reuse at a later time is called a *case* in the AI literature. The ability to "frame" a problem is what differentiates great from ordinary designers. It is the ability to distinguish between the vital few and mundane many design factors that leads to a good design. Kolodner (1993:13) defines a *case* as a contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner. Rather than viewing reasoning primarily as a composition process, *Case-Based Reasoning* (CBR) views reasoning as a process of remembering one or a small set of concrete instances or cases and basing decisions on comparisons between the new situation and the old instance. This view has important implications:

- CBR emphasises the use of concrete instances over abstract operators. It regards large chunks of composed knowledge as the starting point for reasoning. Though there may be smaller and more abstract chunks of knowledge in memory, they derive from cases and are thus secondary to them (Kolodner 1996:361).

- CBR emphasises manipulation of cases over composition, decomposition and recomposition processes. Reasoning by use of cases comes first and composition of operators is of secondary importance (Kolodner 1996:364).

Of all the AI methods available today, Case-Based Design (CBD) is the most promising with regards the storage of previously synthesised design solutions. CBD is a sub-set of CBR that is aimed specifically at design using CBR methods.

CBD facilitates the provision of a comprehensive design database of past solutions that designers will not remember on their own. CBD has distinct advantages over other AI techniques such as *Knowledge-Based Systems* and *Models.*

The CBR paradigm has a bias against problem decomposition and recomposition implied by composition of operators, because composition is a highly complex process. When problems are entirely decomposable into noninteracting parts, decomposition and recomposition are easy. As problems become less and less decomposable into non-interacting parts, recomposition becomes harder and harder. Traditional methods must be stretched beyond their original intent to deal with these problems. Such problems, which are called *barely decomposable*, can be more efficiently solved by methods that do not have to decompose them (Kolodner 1993:16).

Kolodner (1993) is of the opinion that engineering and architectural design is almost entirely a process of adapting old solutions to fit a new situation or merging several old solutions to do the same. Carrara *et al.* (1994) agree with this viewpoint when they characterise design as:

1. Defining a set of functional objectives that ought to be achieved by the design artefact.
2. Constructing design ‘solutions’ which, in the opinion of the designer, are (or should) be capable of achieving the predetermined objectives.
3. Verifying that these solutions are internally consistent and that they achieve the objectives.

Richens (1994:309) strongly disagrees with this point of view when he claims that architectural objectives usually include functional ones, but are dominated by less definable intentions. Flemming (1994:22) states that attempts to introduce machine innovation and creativity are red herrings. Oksala (1994:41) expresses the opinion that it is realistic to design machines that work as architectural design assistants and are capable of redesigning work according to given rules.

It is essential that this database be built up during the normal activities of a design firm. If a designer has generated a solution he should be able to store it literally with the push of a button. Most experimental prototype systems at the moment rely on independent and separate processes that may require the assistance of an expert that is intimately familiar with the technicalities of indexing and retrieval. The author is of the opinion that *Case-Based Systems* would be practical when designers themselves are actively involved in the modification of a case and its storage for re-use. This machine/ designer relationship uses the best of both worlds.

The author argues that CBD is a valid option for the following reasons:

1. Experts (domain experts) using any preferred front-end design method build up the corpus of knowledge. It does not preclude traditional methods. Structured methods would be more efficient.
2. Unlike other structured methods CBR allows a problem to be solved as a complete unit. This is closer to the holistic synthesis of design problems that is dominant in architectural design.
3. Successful precedents in an architectural environment already exist (Kolodner 1993).

If a similar problem has been solved previously, it can provide the glue that holds barely decomposable problems together. Rather than dealing with hard recomposition problems, the reasoner only has to address those parts of the old solution that do not fit the new situation.

Case-Based Reasoning (CBR) is an approach to knowledge, memory structure and reminding that is based upon modelling experiential knowledge. It is characterised in the literature as a problem solving approach of a reasoner, which makes inferences from previous solutions which are adapted to current situations. It has demonstrated its usefulness in domains where experience is strong, but the domain model is weak or poorly formalised (Oxman *et al.* 1994). Rather than duplicating human cognition, these models attempt to capture the essence of the human cognitive processes and to explicate the principles of their operation. A new generation of Knowledge-based System can potentially work in a partnership relationship with the human designer. The objectives of support or aid systems have been defined as to enhance human decision making by suggesting alternatives, predicting consequences and conveniently grouping together the information that goes into decision making.

There are many different types of solutions. The solution to a design problem is the artefact that was designed. With a solution in place, a reasoner that retrieves a case can use its solution to derive a new solution. Solutions also have other components that aid adaptation. The following list from the research community as interpreted by Kolodner (1993:154) is useful:

- The solution itself.
- The set of reasoning steps used to solve the problem. This was well addressed in the AEDES prototype.
- The set of justifications for decisions that were made in solving the problem.
- Acceptable solutions that were not chosen and the reasoning and justifications that go with them.
- Unacceptable solutions that were ruled out and the reasoning and justification that go with them.
- Expectations of the result of deployment of the solution.
- Things that went wrong with the previous solution.

3.2.4.2 Advantages of a Case-Based Reasoner?

CBR has several advantages that give an indication when it should be used. The list below has been collated and adapted from Kolodner (1993). The following advantages can be identified:

1. CBR reasoning allows the reasoner to propose solutions to problems quickly, because it avoids the time necessary to derive those answers from scratch.
2. CBR allows a reasoner to propose solutions in domains that are not completely understood. This is of particular importance to the advanced planning that is necessary to design and build complex facilities such as hospitals.
3. Remembering previous experiences is particularly useful in warning of the potential for problems that have occurred in the past, alerting a reasoner to take action to avoid previous mistakes.
4. CBR can be used as a communication tool between designers and other less design literate participants.
5. Cases help a reasoner to focus his reasoning on important parts of a problem by pointing out what features of a problem are the important ones.
6. A CBR system can be made to learn. In CBR problem solving efforts are saved to expedite future work. Learning is a natural consequence of problem solving efforts. CBR systems can be designed in such a way that they adapt to changes in their environments by means of adaptive fuzzy sets, discussed below. The system can continue to collect cases after deployment.
7. When CBR is used to solve problems, solutions can be justified by the cases they are derived from. In a domain where it is difficult to evaluate solutions objectively such as architectural design, CBR has the advantage of providing illustrations of the effects of particular solutions.
8. CBR can be designed to anticipate potential problems as natural part of their reasoning. Unsuccessful experiences with past solutions can be used in case-based systems to anticipate possible problems that might result from solving a problem a certain way. In general this capability adds efficiency. In architectural design anticipation of problems is critical.
9. CBR provides a way for designers and computers to interact in a realistic way. CBR is fundamentally inspired by human behaviour. Certain tasks in design such as the calculation of energy consumption or acoustic performance is easier for a computer to achieve, whereas aesthetic design decisions is best decided by the designer. Designers are good with creative reasoning, but poor at remembering the full range of applicable cases. Humans tend to be biased in their remembering or as novices they not yet had the experiences they need to solve the problem. During an interview of the professional team involved in a large and complex construction project this fact was emphasised.
10. The knowledge acquisition for a CBR system is natural. Concrete examples rather than piecemeal rules can be used. Experts (experienced practitioners) find it difficult to report the knowledge they use to solve problems. They are quite at home reporting their experiences and discussing the ways in which cases are different from one another.
11. CBR should be considered when it is difficult to formulate domain rules but cases are available. Formulating rules is difficult in weak-theory domains such as architectural briefing and design. In this domain knowledge is incomplete, uncertain or inconsistent. It is impossible to formulate rules when there is a great amount of variability in design situations that have the same outcome.
12. CBR can be considered when rules that can be formulated require more input information that is normally available. This may be due to incomplete specified problems or the fact that the knowledge required is not available at problem-solving time. This is often the case in the construction industry and fast track projects where all project information is not available up-front.

13. CBR should be considered when it is expensive to use rules because the average rule chain is long.
14. CBR should be used when generally applicable knowledge is not sufficient to solve a problem. This could be due to the fact that knowledge changes with context or because some of the knowledge required solving the problem is used only under special circumstances.
15. CBR should be considered when a case library already exists. In the present study some hospital design cases (starter kits) are already available, albeit in an unstructured format.
16. When no fast computation method exists for deriving a solution from scratch, CBR allows new solutions to be derived from old ones. In the case of the basic hospital starter kit set developed at the Division of Building Technology at the CSIR, simple hospitals but different hospital designs can quickly be built by means of different exemplar department and architectural units.
17. When there is no fast computational method for evaluating a solution or when there are so many unknowns that evaluation methods are unusable or difficult to use, CBR provides an alternative.
18. CBR allows evaluation of solutions when no algorithmic method is available for evaluation.
19. Cases are useful in interpreting open-ended and ill-defined concepts.

3.2.4.3 The disadvantages and caveats of Case-Based Reasoning

CBR has several disadvantages and caveats in architectural design that should also be considered. The list below has been collated and adapted from Kolodner (1993):

1. CBR requires cases. Traditionally the effort in building a CBR system went into case collection. It is apparent from a study and interviews[1] with the designers of ARCHIE that it was an enormous effort. To be successful in the architectural profession and the construction industry it should not require such extraordinary efforts. The case library should be automatically assembled during the normal professional design activities.
2. For CBR to be useful and reliable, cases with similar problem statements should have similar solutions. CBR is based on the premise that situations recur in a predictable way. Adaptation modifies old solutions to fit new situations. If a domain is discontinuous where similar situations require wildly different kinds of solutions, then CBR cannot be used and would be misleading. This is unfortunately only partially true in architecture. Creative designers do not always solve related design problems in a similar way.
3. CBR solutions are not guaranteed to be optimal. The full range of possible design solutions is usually not explored in a CBR system intended for design support. Optimal or more creative solutions may be missed. This is a problem in any heuristic system such as TRIZ that is also discussed in the present study. The designer cannot escape his responsibilities, however the CBR system will remind him of design aspects he might have forgotten.
4. An inexperienced case-based reasoner might be tempted to use old cases blindly, relying on previous experience without validating it in the new situation.
5. A case-based reasoner might allow cases to bias him or her too much in solving a new problem.
6. Case libraries require considerable storage space. In the design of CBR systems special consideration must be given to ensure a long life of the case with changing technology. A large sum of money in terms of intellectual capital, time and effort is encapsulated in the case library. Persistence of data is therefore of paramount importance.
7. Inexperienced people are often not reminded of the most appropriate sets of cases when they are reasoning.

---

[1] Janet Kolodner and Craig Zimring personal communication during April 2000.

3.2.4.4 Case-based Reasoning compared with other methods

The CBR/ CBD cycle (Kolodner 1993:18) has striking similarities with the product development method of concept selection proposed by (Pugh, 1996; Ulrich *et al.*, 1995) (Figure 13). In generalised terms the CBD cycle is the case equivalent of concept selection.

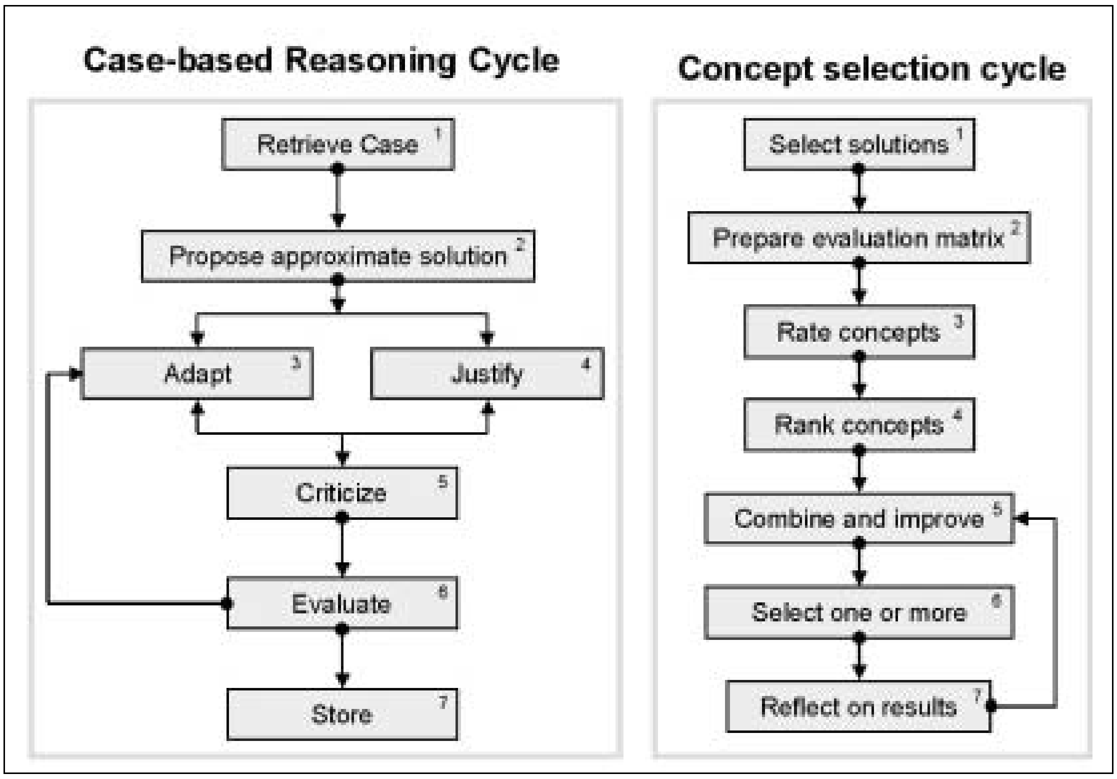


Figure 13: Case-Based Reasoning compared to concept selection (Collated by author from Kolodner (1993:18), Ulrich *et al.* (1995) and Pugh (1996))

The typical stages of the CBD cycle are (Kolodner *et al.* 1996:35):

1. *Retrieval*. Partially matching cases must be retrieved to facilitate reasoning. This is called *case retrieval.* The case was created in the first instance by a *case storage* process also called *memory update*.
2. *Solution proposal*. In problem-solving CBR, a ballpark solution to the new problem is proposed by extracting the solution from the retrieved case.
3. *Adaptation*. This is the process of altering an old solution to fit it to the context of the new situation.
4. *Criticism*. This is a critical analysis of the new solution before applying it.
5. *Justification*. This is the process of creating an argument for the proposed solution, done by a process of comparing and contrasting the new situation with prior cases. Sometimes justification might by followed by a criticism step in which hypothetical situations are generated and the proposed solution applied to them in order to test the solution.
6. *Store (memory update)*. The new case is permanently saved for future use.

The following table compares *Case-Based Reasoning*, (CBR), *Rule-Based Reasoning* (RBR) and *Model-based Reasoning* (MBR)[1].

[1] Janet Kolodner is of the opinion that CBR, MBR and RBR form a continuum. Personal communication 14 April 2000.

Table 4: A comparison between Case-Based, Rule-Based and Model-based Reasoning (Collated by author)

Case-Based Reasoning	**Rule-based Reasoning**	**Model-Based Reasoning**
Cases in case libraries are constants that describe the way things work.	Rules in rule bases are patterns.	Store causal models of devices or domains.
Cases are retrieved that match the input partially.	Rules are retrieved that match the input exactly.	
Cases are retrieved first, approximating the entire solution at once, then adapted and refined to a final answer.	Rules are applied in an iterative cycle of microevents.	
Cases are large chunks of domain knowledge, quite likely redundant, in part, with other cases. Based on idiosyncratic knowledge, specific to episodes but mostly not normative. Provides methods for constructing solutions.	Rules are small, ideally independent but consistent pieces of domain knowledge.	Emphasise general knowledge that covers a domain. Models hold knowledge needed for validation or evaluation of solutions but do not provide methods for constructing solutions.
CBR can be used both when a domain is well and not so well understood. In the latter case it assumes the role of a generalised model.	Not applicable	Is used when a domain is well enough understood to enumerate a causal model.
Provides for efficient solution generation and evaluation is based on the best cases available.	Not applicable	Provides a means of verifying solutions, but solution generation is unguided.
Needs a means of evaluating its solutions, guiding its adaptation and knowing when two cases are similar.	Not applicable	Models provide a means of evaluating its solutions.

These differences led to differences in knowledge acquisition. In RBR, knowledge is extracted from experts and encoded in rules. This is often difficult to achieve. In CBR most (but not all) knowledge is in the form of cases. CBR needs adaptation rules and similarity metrics and more types of knowledge, but knowledge is easier to acquire.

Both MBR and CBR were developed as methods for avoiding reasoning from scratch. Both compose knowledge into large chunks and reason using large chunks. The differences have mostly to do with the content of the knowledge used and the conditions of applicability for each.

3.2.4.5 Types of Case-Based Reasoners

Kolodner (1993) distinguishes between automated reasoners and retrieval-only aiding and advisory systems. Numerous cases can be found in the literature to illustrate the former type that can achieve numerous diverse tasks. Typical examples are:

1. CHEF is a case-based planner. Its domain is recipe creation. Recipes are viewed as plans.
2. CASEY is a case-based diagnostician. It takes as its input a description of its new patient, including normal signs and presenting signs and symptoms. Its output is a causal explanation of the patient’s disorders.
3. JULIA is a case-based designer that works in the domain of meal planning.

4. HYPO is an interpretative reasoner that works in the domain of law. It takes as input a legal situation and as its output it creates an argument for its legal client.
5. PROTOS implements both case-based classification and case-based knowledge acquisition. Given a description of a situation or object, it classifies the situation or object by type.
6. CLAVIER is a manufacturing industry related system for configuring the layout of composite aeroplane parts for curing in an autoclave. It is used at Lockheed in California.
7. ROBBIE (Re-Organisation of Behaviour by Introspective Evaluation) combines a case-based planner with an introspective component. It was used to simulate an intelligent agent travelling in a limited world of a number of street blocks. The agent had to conform to certain basic rules. The agent could intelligently work out alternative rules if an unforeseen obstacle came in the way (Fox 1995).

The former group is where an application in an architectural domain is most likely to achieve success. CBR fits well with the way that designers work. People use CBR naturally in much of their everyday reasoning. Kolodner (1993) provides existing examples that are precedents for the present study. These examples are all retrieval-only aiding and advisory systems. The first one is a hypothetical architect's assistant. ARCHIE and ARCHIE-II are useful precedents of prototype systems that give direction to the present study. ARCHIE-II uses the concept of design stories. Some stories in ARCHIE-II tell about design features that did not work and what could be done to remedy the situation. Others report on features that were successful. Users first describe to the system the problem they are working on. The system subsequently retrieves buildings that are similar to the desired new one. The user can then display the building or part of a building that is retrieved. He is shown a floor plan surrounded by annotations. These annotations describe the various design features.

Domeshek *et al*. (1994) describe the MIDAS (Memory for Initial Design of Aircraft Subsystems) system. This system used ARCHIE as a precedent to support early design of aircraft subsystems. Both ARCHIE-II and MIDAS use the Design-MUSE shell that eases construction of case-based design aids. An important goal of this system was that domain experts should be able to maintain it, rather than AI experts.

Oxman (1994) recognises four cognitive approaches for modelling design case knowledge:

- Generic models (model-based)
- Associative models
- Exemplar models
- Precedent

3.2.4.6 Generic models

A design space is essentially a delineation of a class of things. That is designs conforming to particular meanings and a particular syntax. Knowledge is used to define classes of designs called generic designs. It is often convenient to make the generic nature of knowledge explicit. Rather than using grammatical rules, a design space may be defined in terms of a class description called a *generic model*. This is done by listing all properties of the class, including the ranges of properties that an instance may take and also the interrelationships between properties. A small house can be defined in terms of its generic form and attributes. The graphic structure contains implicit information about the essential properties of the class. The properties could also be listed. An example is the list of allowable rooms it may have. It may also be stated that it includes items such as a roof and a front door.

In some cases a certain design instance is said to typify a class. It embodies the features of its class in which we are interested. Such a design is said to be an *archetype*. The concept of an *archetype* is useful because we prefer to think in terms of instances rather than in terms of the abstract world of classes of things.

In design the term *prototype* is also used. This is generally seen as a design from which other designs originate. A *prototype* typifies a class of designs and serves as a generic design.

3.2.4.7 Associative models

The associative mechanism is another key principle of cognition, which is present in design thinking. In associative reasoning concepts are linked on the basis of conceptual relations to form a structure of concepts. This can be represented by a conceptual network, which maps the structure of relationships and emphasise semantics. A semantic network is the set of all relationships which concepts have to other concepts. The semantic network is related to some context in which it has meaning. This provides the basis upon which to model attribute-based associative thinking in design. In typological design there is a restrictive definition of essential formal variables in the type and how they can be hierarchically modelled into a set of formal concepts. In associative reasoning it is the particular structure of conceptual linkages in contrast to a well-defined hierarchical structure, which is significant.

In architectural design, knowledge associated with recognised categories such as building types provides a clear domain example of typological knowledge through which generic designs can be modelled. With regards to design concepts, there is no comparable consensus on what constitutes the vocabulary of architectural concepts. One area where a vocabulary has begun to emerge is that of the *formal concept*. Formal concepts describe particular features (formal attributes) of the design entities. In the case of architecture, these are the vocabulary of concepts, which describe the formal content of building designs. A system like this could allow for the maintenance, presentation and possible modifications of associative linkages between concepts within the designs.

An example of this type of model is the FORMNET system. This system contains a vocabulary of more than hundred formal attributes that are hierarchically organised into nine major categories and 40 sub-categories. These were established through the survey and analysis of the literature on formal analysis and on the architecture of Le Corbusier. The formal knowledge relative to the villas is organised into a semantic network in which the formal attributes are the nodes. This provides a means to navigate within the system by associative connections between formal concepts, to study the coincidence of formal concepts in various designs and to study relationships between attributes. Some of the attributes that are used in FORMNET are symmetry, grid, regulating lines and free plan. The projects are described both two and three-dimensionally, while the concepts are described two-dimensionally. Historical styles such as Doric and Gothic also provide associative models. Doric gives democratic and Gothic religious associations.

3.2.4.8 Exemplar models

In this approach it is attempted to re-use prior knowledge rather than to generate new designs. The previous solution is adapted to the current situation. Prior knowledge is associated with specific design cases in which the knowledge is highly explicit.

The case, as specific knowledge, can be distinguished from generic knowledge by its unique departure from the norm. A design case has something specific to communicate regarding the solution, its history of generation and its implications in use. Since the knowledge of prior problem solutions is used, the case is structured in such a way that it can be adapted.

Architectural details are an example of this type of case. Building details are example-based and detailing is often based on the re-use of specific examples, which are exemplars, or examples that function as models. The information base could be very broad and special attention has to be paid to the access method. The traditional CI/SfB indexing conventions are not adequate. Organisation of the index according to a convention of typological categories of elements will support conventional search by taxonomic categories (category, element name and product name). However it will not necessarily support search by other categories such as design principles and it will not support browsing and cross-indexing.

Three broad classes of domain knowledge can be identified:

- *Procedural knowledge* is a process or algorithm for design. The design of a staircase is an example where the calculations are based on floor to floor height, length of the stair run, and the tread riser relationships.
- *Causal knowledge* is a detailed procedure for calculation. An example is the calculation and design of partitions for thermal or acoustic properties.
- *Behavioural knowledge* is the understanding of the performance achieved by particular materials or by a particular configuration of elements in a building. This characterises much of the knowledge of building detailing.

Despite the abundance of literature and information in the field, knowledge is generally poorly structured. The knowledge is not structured in such a way that it can be used in models of the design process. It is the integration of knowledge behind the detail within the working environment, which is a long-term objective of intelligent CAD libraries.

Some desirable characteristics of such a system are:

- Memory and indexing approach to support exploration as well as directed search.
- Explanations such as pitfalls and lessons should be integrated into the case.
- The graphic representation should be linked to a model of the case adaptation process.
- Library and the design environments should be integrated.

3.2.4.9 The design precedent

The selection process of relevant ideas from prior designs in current design situations has been termed *precedent-based design*. During the course of exploration of design ideas within precedents, designers are able to browse freely and associatively between multiple precedents in order to make relevant connections. This makes the discovery of unanticipated concepts possible in precedents. In precedent-based systems the ability to encode, search and extract design knowledge relevant to the problem at hand is significant.

One method to represent design knowledge in this type of CBR is to base it upon a decomposition of holistic case knowledge into separate *chunks of design knowledge*. One means to decompose case knowledge into separate and independent chunks is the concept of the story, which is currently employed in the CBR community (Oxman *et al.* 1994:59). The design story is employed as a way to decompose existing descriptions of complex design precedents into chunks. A story is also useful because it provides contextual information. In order to structure a story in a useful format that can be analysed a tri-partite schema that uses an *issue-concept-form* formalism can be used. Each design story is a way to link these three components. Indexing of the cases could become *story indexing* rather than case indexing. Linkages between precedents can be established through matching of issues and design and design concepts.

The design precedent addresses some of the problems of the other models. Because of their network structure, the knowledge representation can use a semantic network or in the form of a node-link structure as provided in hypertext systems.

Precedent-based design is viewed as a significant paradigm in architectural design. However, it has been the subject of less theoretical and research work than typological design. The potential for design aid systems based upon precedent libraries (design thesauri) is another realistic possibility.

3.2.5 Case-based Reasoning indexing and retrieval

3.2.5.1 Introduction

One of the important issues in CBR is retrieval of appropriate cases. The indexing and retrieval methods as described by Kolodner (1993), Flemming (1994) and Charlton *et al.* (1998) already solved the indexing and retrieval problem substantially. It is clear that the indices required to facilitate the initial selection of relevant cases need to be based on linguistic variables. All present methods are based on static linguistic variables that are searched in order to find the most appropriate case. The author is of the opinion that static linguistic variables fail to address the problem of context of the index. An example of this is a description that states that the design for a specific building is *energy efficient*. The linguistic variable *energy efficient* could have been quantified as a design that requires 50 W/m^2. If a significant breakthrough is made in lighting design a new low energy design might be feasible requiring only 20 W/m^2. This would invalidate the previous assumption that 50 W/m^2 is energy efficient. The best solution to this type of problem is to formulate a dynamic, context sensitive linguistic variable *energy efficient*. The value is calculated at the time of retrieval in terms of the known universe of designs. This implies that it is better to store the calculation method with the linguistic variable, rather than absolute values. In the case of lower order values that are absolute such as *gross area*, *rentable area*, *volume* and *reverberation time* it is acceptable to store the values in an absolute way. If static non-linguistic variables are to be compared and weighed then the Flemming method is convenient to weigh up the various factors. The author is of the opinion that Charlton *et al.* (1998:322) comes the closest to a dynamic approach by recommending the use fuzzy sets. However they fail to recognise the need for dynamic linguistic indices.

3.2.5.2 The indexing problem

The *indexing problem* has several parts. When a case is created, appropriate labels must be assigned to ensure that it can be conveniently recalled. Labels are also used at retrieval time to judge the appropriateness of an old case in a new situation. Some of the basic requirements of indices are (Kolodner 1993:194-195):

- They have to anticipate the vocabulary a retriever might use.
- Indexing has to be by concepts that are normally used to describe the items being indexed, whether they are cosmetic features or something more abstract.
- Indexing has to anticipate the circumstances in which a retriever is likely to retrieve something.

Tasks and domains must be analysed to find the functionally relevant descriptors that should be used to describe and index cases. This is called the *indexing vocabulary*. Index vocabulary is a subset of the vocabulary used for full symbolic representations of cases. In the event of retrieval-only CBR it is not necessary to represent the entire contents of the cases symbolically. It is only necessary to represent in the case index the part of the description needed for retrieval. This is called *index assignment*. Indices are those combinations of features of a case that describe the circumstances in which a reasoner might find the case useful during reasoning.

The following general guidelines for choosing indices can be identified (Kolodner 1993:197):

- Indices should be predictive. This is those combinations of descriptors of a case that were responsible for solving it the way it was solved and those combinations that influenced its outcome.
- Predictions that can be made should be useful. They should address the purposes the case will be used for.
- Indices should be abstract enough to make a case useful in a variety of future situations. This often implies that indices should be more abstract than the detail of a particular case.
- Indices should be concrete enough to be easily recognisable in future situations. It should be possible to recognise the case with little inference.

3.2.5.3 Choosing an indexing vocabulary

A vocabulary needs to cover relevant similarities rather than just surface features. Focussing the indexing on the relevant features of a case does this. Because indices are chosen from a case's description, the requirements of the indexing vocabulary are known. The case can be described from two main sets of material (Kolodner 1993:203):

1. The *functional approach*. By means of the *functional methodology* representative domain cases are collected. The corpus of available cases and the tasks that must be supported are examined. For each case the points it can make, the situations in which each point is applicable and the ways the case needs to be described to make it available.
2. The *reminding approach*. The kind of reminding that is natural among human experts who do the designated task is examined. Similarities between new situations and the cases they are reminded of. This is an attempt to find out which descriptors are important to judge similarity and the circumstances.

The indexing vocabulary must capture those domain dimensions that are useful for reminding. One of the attempts to a vocabulary for intentional situations was the Universal Index Frame (UIF). In 1982 Schank proposed organising structures called *Thematic Organisational Packets* (TOPs). TOPs are organisers of cases that are thematically similar to each other. If two cases have the same thematic structure they fall into the same thematic category. The *Universal Index Frame* (UIF) of Schank and Osgood in 1990 built on this concept. It uses the dimensions and vocabulary of goal and plan interactions to structure the descriptions of intentional situations. The UIF suggests the following descriptors or dimensions (Kolodner 1993:228):

- *Anticipatory affect:* the emotions of the character going into the situation
- *Pretask belief:* relevant beliefs of the character going into the situation
- *Task:* the task the actor is actively engaged in as the episode plays itself out
- *Theme:* relevant thematic relationships, roles played by the character, character traits and ambitions that the character brings to the situation
- *Goal:* the character's relevant goal
- *Plan:* the plan the character uses or intends to use in the situation
- *Result:* the major impact of what happened in the situation
- *Positive side effects*
- *Negative side effects*
- *Resultant affect:* the emotions of the character leaving the situation
- *Post-task belief:* relevant beliefs of the character leaving the situation. This is what the character learned from the situation
- *Change in affect:* a characterisation of the degree of change in the character's feelings as a result of the episode

3.2.5.4 Methods for index selection

Kolodner (1993:249) identified the following general steps of index selection:

1. Determine what the case could be useful for.
2. Determine under what circumstances it would be useful.
3. Translate the circumstances into the vocabulary of the reasoner.
4. Synthesise the circumstances to make them as recognisable and generally applicable as possible.

Kolodner (1993:249-281) provides a detailed description of how indices can be chosen. The description below is a summary of these methods:

1. *Choosing indices by hand.* This is used when the cases are complex and the indices need to be accurate. This is also used when the knowledge required to choose the indices accurately is not concretely available or is too complex to insert directly into the computer.
2. *Choosing indices by machine.* This is useful when the problem solving and understanding are already automated. Three methods of automated index selection exist i.e. *checklist-based, difference-based* and *explanation-based* methods.

2.1 Choosing indices based on a checklist

This type of index is based on a specific set of dimensions. The checklist facilitates the process of index selection. For each dimension on the checklist a value is found or computed that describes the case. This method puts a significant responsibility on the system builder, because it is only as good as the previously designed checklist. Typical problems that can be encountered are incomplete checklists that result in insufficient indexing. It is also important to discriminate between important and unimportant dimensions. The following steps summarise the process for setting up a checklist:

- List the tasks that the case retrieval will support.
- For each task, determine the features that tend to predict solutions and outcomes.
- For each kind of feature, compute a set of useful generalisations of the feature. Make sure that the features chosen are recognisable and available during reasoning.
- Create the checklist by collecting the complete set.

The list of heuristics below gives an indication of which features are indexing candidates. Features should chosen that:

- Predict outcomes.
- Be predictive of other features.
- Make the kinds of predictions the reasoner needs.
- Discriminate.

2.2 Difference-based

In this indexing method the purpose of indexing is to keep track of the differences between cases. During retrieval search algorithms can choose the best matching cases from the case library. Not all features that are different across cases make useful indices. To ensure that a difference-based index retriever selects only predictive features, difference-based indexing must be combined with some method of choosing predictive features. One way of achieving this is by a combination of *difference-* and *checklist-based* methods.

As discussed above, *checklist-based* indexing methods focus on which dimensions to focus on for indexing. *Difference-based* methods concentrate on which values along any dimension are useful for indexing. A combination of the two methods allows indexing on predictive dimensions that differentiate a case from other similar ones. The following steps summarise the process to set up this type of index:

- Select a classification for each case.
- Select types of features that are known to be predictive. These are usually context sensitive checklists.
- For each feature its value is computed that results in dimension-value pairs.
- From this list all pairs are removed that are non-predictive in the specific context or normative.

The pairs that are left are those that are predictive and that differentiate the case from others.

2.3 Explanation-based indexing

Difference- and checklist-based indexing methods provide a means of computing predictive features to indices. The problem with this is that indices are based on a model of the features that are usually predictive and do not analyse cases individually for their predictive features. This leads to the problem that features are selected that are not predictive for the particular case or features that are predictive are not indexed.

Explanation-based indexing methods attempt to choose indices appropriately for individual cases. The reasoner uses explanation-based generalisation methods to generalise the explanation. Indices are then chosen from the content of the generalised explanation. In explanation-based indexing, domain knowledge is used to determine which facts of a case are relevant and which can be safely ignored. The index is generalised to the most abstract point where the explanation can still hold. After the reasoner discovers it has made a mistake, it attempts to explain it or assign blame. After explaining the mistake, it extracts from the explanation the concrete recognisable features of the situation that that were responsible for the problem. It then generalises those features to the point where they are still concrete but where the explanation that was derived can still be applied. Unlike checklist-based methods this method chooses as indices only those features that are responsible for the failure. Those features, if observed in future cases, will predict the failure observed in this case. Checklist- and difference-based methods have no means of distinguishing which of the many potentially predictive and differentiating features are responsible for the failure, and will index using far more features.

The explanation-based index selection process consists of the following steps:

1. Create an explanation.
2. Select relevant observable features from the explanation.
3. Generalise those observable features as far as possible, making sure the resulting generalisations are also observable. The original explanation must still apply, given this general description.
4. If the index supports a solution-creation goal, then
    - append additional information specifying the goal the case achieves.
    - generalise the goal appropriately and repeat the process.

3.2.5.5 Retrieving cases from the case library

If a large case design library has been built up it should be possible to conveniently retrieve a case by means of a retrieval procedure. Flemming (1994:84) identified a method that uses the principle of a target index, t. t is compared with all available cases c. Given a target index, the comparison with a case index proceeds object-by-object and attribute-by-attribute for those objects that belong to and those attributes that have values in the target index. In the simplest case, only objects of the same class or type and attributes with the same names are compared. In order to extend the allowed matches several schemes are used that include subtype, subrange and subset matching. The following general rules are defined:

- An object *A* comparable to an object *B* if *B* belongs to the same class or to a subclass of *A*.
- An attribute *a* is comparable to an attribute *b* if *a* is implied by *b*.

Comparability of attributes implies that attributes have values of the same type. For example a minimum x-dimension attribute is implied by a minimum dimension attribute. In the simplest case, the results of comparisons are binary. Sometimes it is important to distinguish whether a lower bound is missed narrowly or by a large degree. It may also be important to know whether only one or several functional units are missing when two *constituent* attributes are compared. Flemming (1994:85) suggests that for each comparison the degree to which it succeeds be computed. This is expressed as the *closed bounded interval* [0,100]. 100 is a perfect match and lower numbers indicate the percentage by which a perfect match has been missed.

When scanning cases for retrieval the individual comparisons must be aggregated so that cases with the best overall fit are presented to the designer. One solution to arrive at ranked values of cases would be to compare the weighed sums of the individual matches. However plausible weights are difficult to determine in building design. Interactions between comparisons cannot be taken into account. The way in which certain deficiencies enter an overall evaluation may depend crucially on the way other comparisons succeed. This is known as the problem of *mutual preferential dependence*. To avoid some of these problems problem features are divided into predetermined priority classes and matches for prioritised features are determined first. Cases that match the most features are preferred. This process uses weighed sums implicitly. Features in the same class are given the same weight and matches are added up. Features in higher classes overrule those in lower classes. The designer is able to decide which features he is going to search on.

The method of calculation can be formalised in the following way by defining a special retrieval function

$$\varphi(t, c)$$

which returns a real number in the *closed bounded interval* [0,100] to express the match between a target index t and a case index c.

φ unpacks t recursively in terms of its valued attributes and computes their matches with comparable valued attributes in c.

φ is specialised with regards the objects or data types that have to be compared. Some special forms of φ are indicated below. The subscripts indicate the type of specialisation.

$$\varphi_{OBJ}(A,B)=\begin{cases} 0 & \text{if } A \text{ and } B \text{ are not comparable;} \\ 100 & \text{if } A \text{ has no valued attribute;} \\ \sum_a w_a \varphi_{ATTR,OBJ}(a,B) & \text{otherwise,} \end{cases}$$

Unordered lists like the value of attribute tests can be compared similarly to φ_{OBJ}. The sum goes over all valued attributes a of A. w_a are weights with

$\sum_a w_a = 1$; and

$$\varphi_{ATTR,OBJ}(a,B)=\begin{cases} \varphi_{ATTR}(a,b) \text{ if } B \text{ contains a valued attribute } b \text{ comparable with } a\text{;} \\ 0 \text{ otherwise} \end{cases}$$

$\varphi_{ATTR}(a,b)$ is specialised with respect to the data type of a and b. For attributes whose data types are lower bounds (like the attributes *minimum-width* and *min-area* of a functional unit of variable size.

$$\varphi_{LOWER\_BOUND}(a,b)=\begin{cases} 100\ [b]/[a] & \text{if } [b]<[a]; \\ 100 \text{ otherwise.} \end{cases}$$

Where $[x]$ denotes the value of an attribute x. Upper bounds as well as general numbers can be treated similarly. Names match either completely (100) or not at all (0).

The basic form of φ that unravels a *constituent* attribute in a target index can be defined as follows:

$$\varphi_{CONST\_LIST}(K,L)= \max \sum_k w_k \varphi_{OBJ}[k,\gamma(k,L)]$$

where the sum goes over all objects k in K. $\gamma(k,L)$ is defined as an operator that traverses L and the constituents of the objects in L to grab a corresponding object comparable with k. γ satisfies the following conditions:

1. If no corresponding object can be found γ returns a dummy object to enforce a 0 value of φ for this particular k.
2. Repeated calls to γ will not return objects that have been returned before; that is, the mapping established by γ from objects in K to objects in L and their constituents is right-unique.
3. If γ maps an object in K to a constituent m of an object l in L, it does not map other objects in K to objects on the path from l to m.

Charlton *et al.* (1998:324) state that the descriptions on which retrieval of a relevant case depends are ultimately based on classifications. A classification consisting of restricted values is seen as a flat classification. The very reason for the existence of a classification is to enable stored cases to be retrieved. The methodology discussed above is not very useful at the level where the decision must taken if a design case is appropriate at all for the design problem under consideration. This methodology is more appropriate at a direct technical level where various different technical factors need to be directly considered before a final conclusion is reached. It is difficult to develop suitable indices, because it needs to consider the reference framework of the user. Static indices are rigid because they are unable to adapt to the context of use.

Charlton *et al.* (1998:322) suggested the use of static fuzzy sets with labels that are meaningful to the designer. For each prototypical case, designers are asked to specify its membership values in fuzzy sets. Each label specifies the degree to which the particular artefact is part of the fuzzy set identified by the label's name. Collectively, the labels can be seen as providing a multitude of descriptive names for a case, instead of a single possibility. The use of fuzzy sets as described by Charlton is more flexible than Flemming's, however the membership values in the fuzzy sets themselves are still static. The author is of the opinion that this can be significantly improved by means of a method of dynamic fuzzy sets. A description of this proposed methodology follows below.

3.2.5.6 The use of fuzzy sets for case indexing

It is now 35 years since the first creation of Fuzzy Sets and Fuzzy Logic that bridge mathematical precision and the vagueness of common-sense reasoning. Fuzzy sets have been successfully implemented in numerous commercial products such as vacuum cleaners, washing machines, rice cookers and cameras that resulted in energy efficiency and increased convenience for the consumer. The Japanese city of Sendai has been using subway trains controlled by fuzzy logic since 1986.

Bellman and Zadeh (1970) and Bojadziev *et al.* (1995:113) describe fuzzy sets as a special class of object in which there is no sharp boundary between those objects that belong to the class and those that do not. Below follows a short summary of the main characteristics of fuzzy sets.

(1) Let $A = \{(x, \mu_A(x)) \quad | \quad x \in X, \mu_A(x) \in [0,1]\}$

Where $\mu_A(x)$ is a function called the *membership function* of *x in A.* $\mu_A(x)$ and $\mu_A : X \to M$ is a function from X to a space called the membership space. When M only contains two points, 0 and 1, A is nonfuzzy and its membership is identical with the characteristic function of a nonfuzzy set. It can be assumed that M is the closed interval [0,1], with 0 and 1 representing respectively the lowest and highest grades of membership.

A fuzzy set is *normalised* when at least one $x \in X$ attains the maximum membership grade 1, otherwise the set is called *non-normalised*. Assume that the set *X is non-normalized,* then *max* $\mu_A(x) < 1$. To normalise the set *X* means to normalise its membership function $\mu_A(x)$. This is given by:

$$\frac{\mu_A(x)}{\max \mu_A(x)}$$

Empty set. A is called an empty set labelled ϕ if $\mu_A(x) = 0$ for each $x \in A$.

Fuzzy singleton. The fuzzy set $A = \{(x_i, \mu_A(x_i))\}$, where x_i is the only value in $A \subset U$ and $\mu_A(x_i) \in [0,1]$.

α *- level set* or α *-cut.* This is denoted by A_α and is the crisp set of elements which belong to A at least to the degree α :
$A_\alpha = \{x \mid x \in U, \mu_A(x) \geq \alpha\}, \alpha \in [0,1]$

Strong α *- level set.* This is defined by:
$A'_\alpha = \{x \mid x \in U, \mu_A(x) > \alpha, \}, \alpha \in [0,1]$

In many practical situations the membership function μ_A has to be estimated from partial information about the subject of consideration. The problem of estimating μ_A from the knowledge of the set of pairs $(x_1, \mu_A(x_1)),....,(x_N, \mu_A(x_N))$ is the problem of abstraction. This problem plays a central role in pattern recognition, but also in the selection of a suitable case for architectural design. Similar abstractions had to be made to calculate the average condition and suitability of facilities during the National Health Facilities Audit (NHFA) in South Africa. In the case of the NHFA criteria had to be carefully derived to determine what the rating of a particular construction element should be on a scale of [1,5] where 5 denoted the optimum and 1 the worst case scenario. In the internal calculations this was replaced by a normalised *closed bounded interval* [0,1].

Equality. Two fuzzy sets are equal, written as $A = B$, if and only if $\mu_A = \mu_B$. That is $\mu_A(x) = \mu_B(x)$ for all x in X .

Containment. A fuzzy set A is contained in or is a subset of fuzzy set B, written as $A \subset B$, if and only if $\mu_A \leq \mu_B$. The fuzzy set of energy efficient buildings is a subset of the fuzzy set of buildings.

Complementation. A' is said to be the complement of A if and only if $\mu_A' = 1 - \mu_A$. For example, the fuzzy sets: $A = \{\textit{high\_rise\_buildings}\}$ and $A' = \{\textit{not high\_rise\_buildings}\}$ are complements of one another if the negation “not” is interpreted as an operation which replaces $\mu_A(x)$ with $1 - \mu_A(x)$ for each x in X.

Intersection. The intersection of A and B is denoted by $A \cap B$ and is defined as the largest fuzzy set contained in both A and B. The membership function of $A \cap B$ is given by

(2) $\mu_{A \cap B}(x) = Min(\mu_A(x), \mu_B(x)), x \in X$ where $Min(a,b) = a$ if $a \leq b$ and $Min(a,b) = b$ if $a > b$. In infix form, using the conjunction symbol ϖ in place of Min, (2) can be written more simply as

(3) $\mu_{A \cap B} = \mu_A \varpi \mu_B$. The notion of intersection bears a close relation to the connective “and”. If A is the class of high rise buildings and B is the class of energy efficient buildings, then $A \cap B$ is the class of buildings that is both high rise and energy efficient. It should be noted that in the example “and” is interpreted in a “hard” sense. That is, we do not allow any trade-off between $\mu_A(x)$ and $\mu_B(x)$ so long as $\mu_A(x) > \mu_B(x)$ or vice-versa. For example if $\mu_A(x) = 0.8$ and $\mu_B(x) = 0.5$, then $\mu_{A \cap B}(x) = 0.5$ so long as $\mu_A(x) \geq 0.5$. In some cases, a softer interpretation of “and” which corresponds to forming the algebraic product of $\mu_A(x)$ and $\mu_B(x)$, rather than the conjunction $\mu_A(x) \varpi \mu_B(x)$ may be closer to the intended meaning of “and”. From the mathematical as well as the practical point of view, the identification

of “and” with ϖ is preferable to its identification with the product, except where ϖ clearly does not express the sense in which one wants “and” to be interpreted.

Union. The union of A and B is denoted by $A \bigcup B$ and is defined as the smallest fuzzy set containing both A and B. The membership function of $A \bigcup B$ is given by

(4) $\mu_{A \cup B}(x) = Max(\mu_A(x), \mu_B(x)), x \in X$ where $Max(a,b) = a$ if $a \geq b$ and $Max(a,b) = b$ if $a < b$. In infix form, using the disjunction symbol ω in place of Max, (4) can be written more simply as

(5) $\mu_{A \cup B} = \mu_A \, \omega \, \mu_B$.

As in the case of the intersection, the union of A and B bears a close relation to the connective “or”. If A = {*high_rise_buildings*} and B = {*energy_efficient_buildings*}, then $A \bigcup B$ = {*high_rise_buildings* or *energy_efficient_buildings*}. As in the case mentioned above a “hard”, “or” which corresponds to (5) and a soft “or” that corresponds to the algebraic sum of A and B can be distinguished. This latter is denoted by $A \oplus B$ and is defined by (7).

Algebraic product. The algebraic product of A and B is denoted by AB and is defined by

(6) $\mu_{AB}(x) = \mu_A(x)\mu_B(x), x \in X$.

Algebraic sum. The algebraic sum of A and B is denoted by $A \oplus B$ and is defined by

(7) $\mu_{A \oplus B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x), x \in X$.

From (7) it follows that

(8) $A \oplus B = (A' B')'$.

ϖ and ω are associative and distributive over one another. (product) and ⊕ (sum) are associative but not distributive.

3.2.5.7 The use of fuzzy sets to formulate dynamic linguistic variables for case retrieval

Variables whose values are words or sentences in natural or artificial language are called *linguistic variables*. Natural language words is a convenient means to retrieve architectural design cases, because humans think in terms of words that most closely describe the desired design qualities. To illustrate the concept of a *linguistic variable* consider the word *age* in a natural language. The meaning of this word is the summary of an enormous large number of individuals. It cannot be characterised precisely. This word also has a different meaning in different domains. The meaning of *age* in a building domain is something totally different to *age* in a human context. The discussion will continue with *age* in the context of buildings. By means of fuzzy sets *age* can be described more precisely. *Age* is a linguistic variable consisting of fuzzy sets such as *very_new*, *new*, *old* and *historic*. These words are called terms of the linguistic variable *age*. Each term is defined by an appropriate membership function. Bojadziev (1995:178) states that good candidates for membership functions are triangular, trapezoidal or bell-type shapes with or without a flat. These mathematical shapes describe the different ways membership functions can be structured. An example is a triangular fuzzy number that are very often used in applications such as fuzzy controllers, managerial decision making and the social sciences. The underlying advantage of the fuzzy relationship shapes mentioned is that membership functions for terms using them can be constructed on the basis of little information.

Let us describe the linguistic variable *age* on the universal set $U = [0,200]$ (Figure 14) by means of triangular fuzzy numbers, which specify the terms *very_new*, *new*, *old* and *historic*.

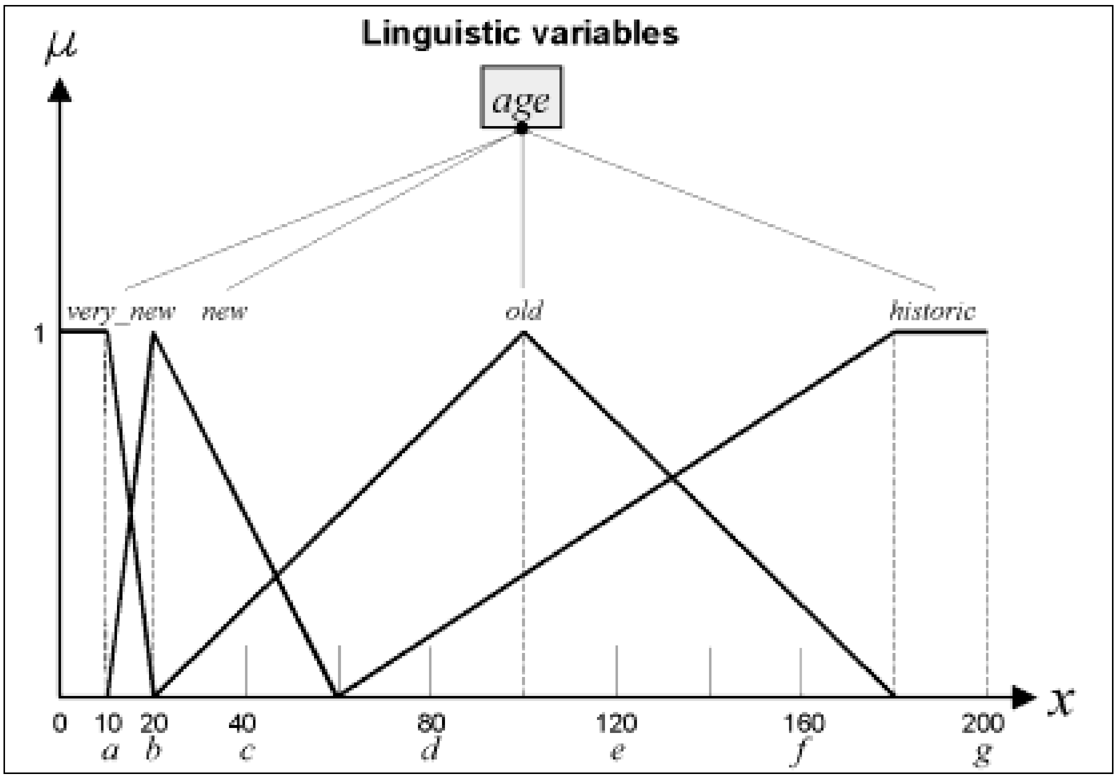


Figure 14: Terms of the linguistic variable *age* in a building context (Author)

The membership functions of the terms using a triangular calculation are:

$$\mu_{very\_new}(x) = \begin{cases} 1 & \text{for} \quad 0 \le x \le 10, \\ \dfrac{20-x}{10} & \text{for} \quad 10 \le x \le 20, \end{cases}$$

$$\mu_{new}(x) = \begin{cases} \dfrac{x-10}{10} & \text{for} \quad 10 \le x \le 20, \\ \dfrac{60-x}{40} & \text{for} \quad 20 \le x \le 60, \end{cases}$$

$$\mu_{old}(x) = \begin{cases} \dfrac{x-20}{80} & \text{for} \quad 20 \le x \le 100, \\ \dfrac{180-x}{80} & \text{for} \quad 100 \le x \le 180, \end{cases}$$

$$\mu_{historic}(x) = \begin{cases} \dfrac{x-60}{120} & \text{for} \quad 60 \le x \le 180, \\ 1 & \text{for} \quad 180 \le x \le 200, \end{cases}$$

Note that the triangular μ values for the linguistic terms of *age* are not linear. In this case it can also be seen that compression of scale occurs at the *very_new* end. An important limitation, of linguistic terms defined like these in the example, is that they are static. This implies that the structure is not self-adjusting if the context where the terms that are used changes. This limits the universal application of terms that are defined in this way. The author proposes a system of linguistic variables to be defined that stores the calculation method. When the linguistic variable is brought into a specific context, then the terms would assume the correct relative values in the context of the specific environment.

Linguistic variables are important in applications. The parameters of technical systems such as condition, suitability, energy, temperature, weight, speed, pressure and heat can be understood as linguistic variables.

3.2.5.8 Fuzzy set linguistic modifiers

Let $x \in U$ and A is a fuzzy set with membership function $\mu_A(x)$. Assume that m is a *linguistic modifier* such as *very*, *not* and *fairly*. mA is a modified fuzzy set whose membership function $\mu_{mA}(x)$ is a composition of a suitable function $f(x)$ and $f(\mu_A(x))$.

The following selections for $f(x)$ are often used to describe the modifiers *not*, *very* and *fairly*.

$f(x) = 1 - x$ not, $\mu_{notA}(x) = 1 - \mu_A(x),$

$f(x) = x^2$ very, $\mu_{veryA}(x) = [\mu_A(x)]^2,$

$f(x) = x^{\frac{1}{2}}$ fairly, $\mu_{fairlyA}(x) = [\mu_A(x)]^{\frac{1}{2}}.$

Consider the fuzzy set A that describes the size of a particular facility in terms of gross m² by the linguistic variable *SIZE*. Assume a small database of five facilities each having a specific gross m².

Facility Name	Facility Code	Gross m²
Facility 1	F_1	5 830
Facility 2	F_2	1 431
Facility 3	F_3	12 979
Facility 4	F_4	11 500
Facility 5	F_5	7 500

Assume further that *SIZE* has three terms *large*, *medium* and *small* having the following values in the *closed bounded interval* [0,1].

large	[0.66,1]	$0.66 \le x \le 1$
medium	[0.33,0.66)	$0.33 \le x < 0.66$
small	[0,0.33)	$0 \le x < 0.33$

In order to ensure a dynamic and flexible system the values of the terms are expressed in terms of a universal set in the context of the application under consideration. The gross m² values therefore range from $[Min(F_{area}), Max(F_{area})]$ where F_{area} is the facility gross area.

In terms of the small example database above the range of values from *small* to *large* would be:

(1) [1431,12979]

Assume that Facility F_x has a gross m² area of 8 000 m². In terms of the data above it can be stated that:

(2) $$\mu_{size}(x) = \frac{F_{x\_area}}{(Max(F_a) - Min(F_a))} = 0{,}69 = large$$

F_{x_area} is the area of Facility *x*.

Assume that a new facility with a gross area of 15 000 m² is added to the database above. In terms of Facility F_x the following is now true:

(3) $$\mu_{size}(x) = \frac{F_{x\_area}}{(Max(F_a) - Min(F_a))} = 0{,}59 = medium$$

Due to the inclusion of the large facility, F_x has been reclassified as *medium*. Due to the flexible formulated definitions the system under consideration will be self-adjusting. This is especially useful when fuzzy sets are considered that give a measure of performance such as energy use.

In the case of interpreting case indices the *intersection*, *union* and *complement* are the most useful. The author is of the opinion that fuzzy sets can be used to select the most appropriate cases from the possible set of cases in the CBR based system envisioned. If a vocabulary of words can be carefully selected that best describe certain index phenomena then, by means of a process of abstraction suitable μ_A, values can be allocated.

The author comes to the conclusion that the calculations that were made in the NHFA with regards condition and suitability are really a subset of total number of possible fuzzy sets possible in this domain. To quantify condition in abovementioned audit the author allocated discrete meanings to fuzzy condition rating words such as *as_new*, *maintain*, *repair*, *replace/ upgrade* and *condemn/ leave*. Each of these keywords was allocated a value in the [0,1] range:

as new	*= 1,0*
maintain	*= 0,8*
repair	*= 0,6*
replace/ upgrade	*= 0,4*
condemn/ leave	*= 0,2*

In a similar way suitability assessments were allocated keywords with values in the [0,1] range:

ideal	*= 1,0*
acceptable	*= 0,8*
tolerable	*= 0,6*
hardly tolerable	*= 0,4*
intolerable	*= 0,2*

The contribution of cost per gross m² for each of 98 construction elements was derived from an analysis of bills of quantities from quantity surveyors. During the audit the average condition for a particular facility was derived by means of the following:

(1) $\sum_{e=1}^{98} w_e ca$ where the meaning of the variables is:

$e =$ *the construction element number*

$w_e =$ *the condition cost model weight in the range* $[0,1]$

$c =$ *condition rating in the range* $[0,1]$

$a =$ *gross area in m² of the department, building or total facility are where the element occurs.*

In a similar way the average suitability for the facility was calculated as:

(2) $\sum_{e=1}^{98} w_e sa$ where the meaning of the variables is:

$e =$ *the construction element number*

$w_e =$ *the suitability cost model weight in the range* $[0,1]$

$s =$ *suitability rating in the range* $[0,1]$

$a =$ *gross area in m² of the department, building or total facility are where the element occurs.*

Abovementioned calculations resulted in the average condition and suitability per facility that could be summarised to district, region, province and country level. The conclusion is made that on the basis of abovementioned concept super concepts can be defined that would facilitate the powerful manipulation of derived high level concepts. The processing domain of the facilities audit is limited. This makes it feasible to formulate fuzzy data abstractions. The keyword *SIZE* can be expressed in terms of the size in gross *m²* found in the audit. It is impossible to define *SIZE* as a universal quantified concept.

An example of an expression in fuzzy terms could be the following:

List all *large* hospitals (size description), in a *new* condition that occur in the province of *Western Cape* (location). In this case we have two fuzzy variables and one non-fuzzy variable. If suitable ranges of words (labels) can be defined and by means of abstraction fuzzy values be allocated then the fuzzy operations, *intersection* and *union* can be used.

Structured Query Language (SQL) as implemented in Oracle has the capability to process sets, although this capability is not often used. The following SQL operators are available in the SELECT statement:

UNION	*Combines two queries and returns all distinct rows returned by either individual query.*
UNION ALL	*Combines two queries and returns all rows returned by either query, including duplicates.*
INTERSECT	*Combines two queries and returns all distinct rows returned by both individual queries.*
MINUS	*Combines two queries and returns all distinct rows by the first but not by the second.*

These set operators make it possible to implement traditional set theory easily. This can readily be expanded to implement fuzzy sets. The following would be required:

- Definition library (labels) of concepts that will typically be manipulated.
- Standard functions and procedures that can be included in a CBR program or object.
- A data set where the domain operational parameters are known.

The following list of linguistic fuzzy sets and terms can be defined for use in queries related to the life cycle of buildings:

CONDITION

new	*= 1,0*
maintain	*= 0,8*
repair	*= 0,6*
replace	*= 0,4*
condemn	*= 0,2*

SUITABILITY

ideal	*= 1,0*
acceptable	*= 0,8*
tolerable	*= 0,6*
hardly tolerable	*= 0,4*
intolerable	*= 0,2*

SIZE

very large
large
average
small
very small

AGE

very old
old
recent
new

DISTANCE

far
close

LARGE

exceptionally large
very large
large
average size

SMALL

very small
small

average size

UTILISATION

totally over utilised
very over utilised
over utilised
normal use
under utilised
significantly under utilised
under utilised

In all cases the keyword that is closest to the main subject in the list appears at the top of the list. These keywords can be converted into static fuzzy set labels by allocating approximate membership values. It can be assumed that μ_A will be in the range [0,1]. In the case of CONDITION the values could be:

new	$0.9 \leq \mu_A(x) \leq 1.0$
maintain	$0.7 \leq \mu_A(x) < 0.9$
repair	$0.5 \leq \mu_A(x) < 0.7$
replace	$0.3 \leq \mu_A(x) < 0.5$
condemn	$0.0 \leq \mu_A(x) < 0.3$

In this case all the values are linear. The abstraction to the particular values was calculated in such a way as to get a clear distinction between the different condition categories in order to map it to colours. In this case absolute accuracy was not important. It doesn't matter how many terms the particular fuzzy set contains. If the fuzzy set only contains two categories then it becomes a traditional set.

If a user needs to define a dynamic fuzzy set that displays a list of all *new* (condition) buildings, that is *small* (gross area) the equivalent dynamic database SQL statement could be the following:

```
SELECT FacilityCode
FROM FacilityResource
WHERE (RemainingResource/TotalResource) >= 0.9
AND ConditionCode = 0

INTERSECT

SELECT FacilityCode
FROM Facility
WHERE ((FacilityArea/(MAX(FacilityArea) – MIN(FacilityArea)) < 0.4)
AND ((FacilityArea/321300.0) >=0.2)

INTERSECT

SELECT FacilityCode
FROM Facility
WHERE FacilityCode LIKE ‘WCP%’;
```

3.2.6 Conclusion

Kolodner (1993:263) suggests that the following methods be used to maintain context sensitivity in the case index selection.

- Use several checklists, each organised around a different well-known context.
- Keep track of how useful individual indices are and modify lists when they are not useful.

Kolodner (1993) also suggested the method of parameter adjustment for interpolating values in a new solution based on those from an old one. In parameter adjustment changes in parameters in an old solution are made in response to differences between problem specifications in an old and a new case. Several case-based reasoning systems use parameter adjustment as a method of adaptation. A system called PERSUADER adjusts old labour-management contracts with new information. If an old contract was signed in a location where the cost of living is high and has risen faster than the norm, but it is not the case in the new dispute, then a smaller percentage wage increase is in order in the new contract.

In all cases the use of a dynamic adaptive fuzzy set based indexing system comes the closest in solving the problems associated with context sensitivity and parameter adjustment. The inherent flexibility of fuzzy sets make them ideal for indexing in many different environments as well as level of detail. This will be the case with design cases found in the construction industry.

Linguistic variables offer a convenient means to intensify or soften the effect of the terms of a fuzzy set.

3.3 The systems view of the world

3.3.1 Introduction

In this section manufacturing, concurrent engineering, Taguchi techniques and the Fuzzy Front End is included because of the prominence of these in the world of manufacturing. Architectural design is seen as a type of low-quantity production. Concurrent Engineering attempts to speed up the engineering process in order to be more effective. Taguchi Techniques indicate how big the impact of small variations in critical parameters or dimensions might be. The Fuzzy Front End provides an opportunity to buy value time during the design process.

In this study it is proposed that architectural design experience be packaged in cases and design starter kits. A case is seen as an entire project where all the design knowledge is stored in the form of artefact descriptions and process descriptions. Artefact descriptions consist of shape- and functional views. The process description consists of sequences over time. The author observed that the data that are required to structure the existence of an artefact over the life cycle exist in different worlds spread over time. These worlds were identified in chapter 3.2. It is observed that attempts to unify the different worlds into one single model such as the Industry Foundation Classes, discussed under 3.2 is unlikely to succeed. In the author's experience it is far more flexible to use processes as a means of formulating relationships between these worlds over time. This has been tested in the PREMIS facilities management system.

The present study attempts to store architectural design knowledge in the form of cases and to create portable mini architectural design cases (starter kits) which many domain-specific tools such as CAD and spreadsheets can share. The term mini design case refers to a design case at a level where it becomes portable and small enough to plug into many different design environments. It must also be small enough to be conveniently distributable via the World Wide Web. This approach is supported by Charlton *et al.* (1988:311) where a Common Product Data Model (CPDM) is mentioned. In the CPDM design data is represented by structured objects, which can be shared. The CPDM can capture a large portion of the data involved in the artefact's development without coercing artefacts into static class hierarchies. This allows flexible and dynamic modelling in terms of multiple object perspectives, dynamic object reclassification and dynamic class evolution. Attempts to achieve this were made in the prototype system *AEDES*, however the integration between the artefact description and process description is still primitive.

Once a user has decided to use a specific mini design case it will be brought into a specific context. The purpose of this chapter is to explore the fundamental characteristics of the context of the project environment that consists of main topics such as processes, product modelling and life cycle decision validation. In a highly competitive environment the processes and product modelling could be concurrent. Although there are strong similarities between the construction and manufacturing industry there are also fundamental differences.

It is important to realise that the best techniques would not succeed if the motivation, attitude, spirit, personality are not supported by the corporate culture.

3.3.2 What is manufacturing?

The word *manufacture* is derived from two Latin words *manus* (hand) and *factus* (made). The combination means made by hand. Modern manufacturing is accomplished by automated and computer-controlled machinery that is manually supervised. Manufacturing can be defined in many ways with two directly applicable to the manufacturing industry (Groover 1996):

- Technologically.
- Economically.

Other types of manufacturing not addressed in this study are:

- Energy manufacturing.
- Environmentally.
- Informatically.
- Socially.

Technologically, manufacturing is the application of physical and chemical processes to alter the geometry, properties and appearance of a given starting material to make parts or products. The processes to accomplish manufacturing involve a combination of machinery, tools, power and manual labour (Figure 15). Manufacturing is almost always carried out as a sequence of operations. Each operation brings the material closer to the final desired state.

Economically, manufacturing is the transformation of materials into items of greater value by means of one or more processing and/or assembly operations (Figure 15). Manufacturing adds value to the material by changing its shape or properties or by combining it with other materials that have been similarly altered.

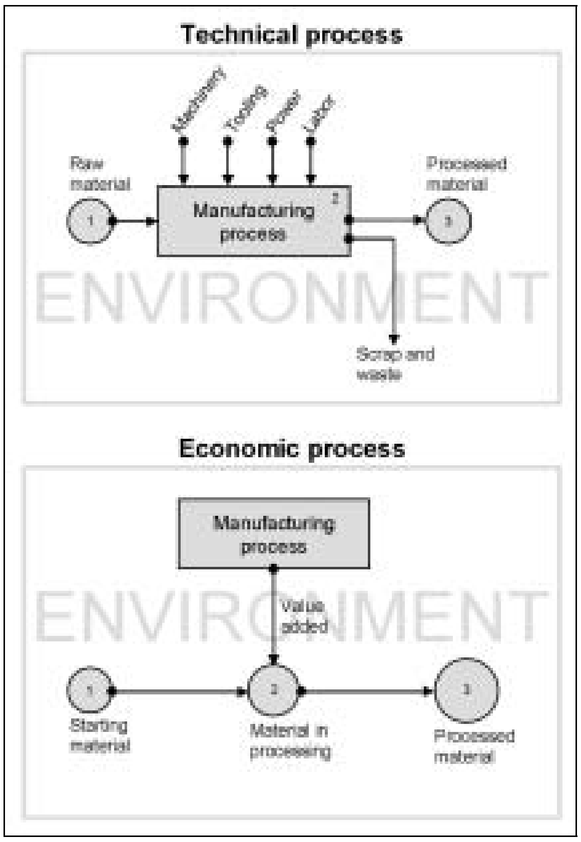


Figure 15: Two ways to define manufacturing, a technical or an economic process (Groover 1996:3)

Groover (1996) identifies *primary*, *secondary* and *tertiary* industries. *Primary industries* are those that cultivate and exploit natural resources such as agriculture and mining. *Secondary industries* take the outputs of the primary industries and convert them into consumer and capital goods. Manufacturing is the principal activity in this category, but it also includes construction and power utilities. *Tertiary industries* constitute the service sector of the economy.

The quantity of products made by a factory has an important influence on the way its people, facilities and procedures are organised. Production quantity refers to the number of units produced annually of a particular product type. Product variety refers to different product designs or types that are produced in the plant. The construction industry is presently a low quantity high variety industry. There is an inverse correlation between product variety and production quantity in terms of factory operations. If a factory's product variety is high, then its production quantity is likely to be low. If the production quantity is high, then product variety will be low. The terms soft and hard product variety can be identified. *Soft product variety* occurs when there are only small differences between products, such as the differences between car models made on the same production line. In an assembled product, soft variety is characterised by a high proportion of common parts among the models. In hard product variety, the products differ substantially and there are few common parts, if any. Again the construction industry is unique in the sense that a lot of parts are common at a low level, but a large variety exist at higher levels. There is also variation between the various construction trades as to the level of standardisation that can be achieved. Air-conditioning parts can be standardised and pre-assembled in a factory before being brought onto site, however this is less feasible with structural elements such as slabs, columns and walls.

3.3.2.1 Manufacturing capability

Manufacturing plants consist of *processes* and *systems* designed to transform a certain limited range of *materials* into products of increased value. The three building blocks, materials, processes and systems constitute the subject of modern manufacturing. There is a strong interdependence among these factors. A company engaged in manufacturing cannot do everything. *Manufacturing capability* refers to the technical and physical limitations of a manufacturing firm and each of its plants. The following dimensions of this capability can be identified:

- Technological processing capability.
- Physical size and weight of product.
- Production capacity.

3.3.2.2 Manufacturing processes

Manufacturing processes can be divided into two basic types:

- Processing operations.
- Assembly operations.

A processing operation transforms a work material from one state of completion to a more advanced state that is closer to the final desired product. It adds value by changing the geometry, properties or appearance of the starting material. An assembly operation joins two or more components in order to create a new entity, which is called an assembly or sub-assembly.

3.3.2.3 Low-quantity production

Groover (1996:21) describes this type of production as a low-quantity range of 1 to 100 units/year. The construction industry bears a close resemblance to this type of manufacturing. In manufacturing the term job shop is often used to describe this type of production facility.

A job shop makes low quantities of specialised and customised products. The products are typically complex, such as space capsules, prototype aircraft and special machinery. Construction activities are normally not nearly as complex as the former.

A job shop must be designed for maximum flexibility in order to deal with the wide product variations encountered. In an analysis by the author of a large construction project this is evident in the large variation of project team configurations and types of construction projects undertaken. If the product is large and heavy and difficult to move in the factory, it typically remains in a single location during its fabrication or assembly. Workers and processing equipment are brought to the product, rather than moving the product to the equipment. Examples of such products include ships, aircraft, railway locomotives and heavy machinery. These products are usually built in large modules at single locations and then the completed modules are brought together for final assembly using large-capacity cranes. In South Africa these practices are not widespread in the construction industry and in-situ construction predominates.

3.3.3 Concurrent engineering (CE)

Many terms have been used to describe similar approaches, including simultaneous engineering, life cycle engineering, design integrated manufacturing, design fusion, early manufacturing involvement, parallel engineering, concurrent design and design in the large.

Ziemke *et al*. (1993:26) trace the origins of CE back to 1940, during the Second World War. The American Aviation Corporation received an order for 320 NA-73 fighter aircraft from the British Air Purchasing Commission. These aircraft were later known as the US P-51 Mustangs. The condition of this order was that the first prototype, NA-73X, had to be ready for testing 120 days after receipt of contract. Given the short schedule one would have assumed that the design engineers would only have used proven and conservative design features. Instead the Mustang included the first use of novel concepts such as laminar flow airfoils and the introduction of a combined radiator housing-ejector nozzle that provided 300 pounds of jet thrust, instead of the usual radiator air drag. The aircraft was designed and built in 102 days. During that time, 2 800 drawings representing 600 000 hours of effort were produced. In retrospect it now seems that critical success factors in such wartime design and development teams were their small size, their broadly experienced leadership and above all, motivation.

Currently there is a different reason for CE. During the last decade the life cycle time of products from different branches of industry decreased while the time spent on product development greatly increased. This is known as the time-trap. In terms of the local construction industry this is an over-simplification, because other factors such as the period of high inflation and the fact that buildings are still constructed for a relatively long life. In the new Menlyn Shopping centre project the planning horizon is 25 years. Due to these changes the pay-off period between market entry and amortisation extended as well. In order to meet the challenges of successfully competing in innovative markets the development and design of new products has become one of the most significant factors. The situation can be characterised by three main tendencies:

- Shift from a seller’s to a buyer’s market.
- Increasing globalisation.
- Change in the importance of technology.

The optimisation of the magic triangle that consists of time, quality and costs is necessary to face competition and complexity in the changed environment described above.

The most important elements when applying CE are people and the design of product development processes. Co-operation and communication are regarded as the most important success factors by companies, which are successfully practising CE. This involves:

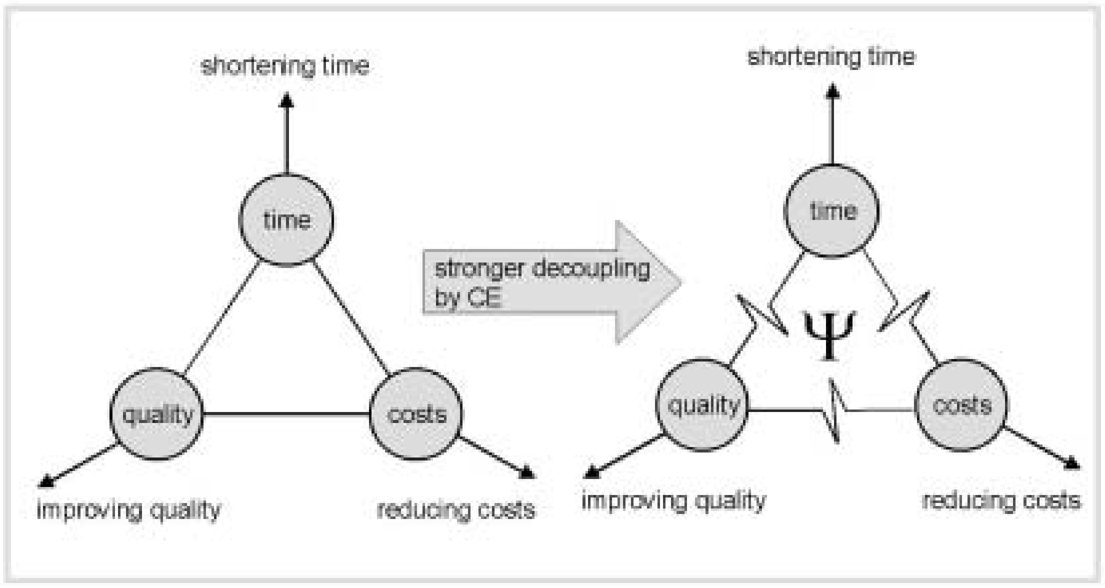


Figure 16: Decoupling of time, cost and quality by means of Concurrent Engineering (Berndes 1996)

- Cutting back barriers among departments and hierarchies.
- Promoting interdepartmental co-operation.
- Building up close links between suppliers and customers.
- Support of CE by top management.

3.3.3.1 Strategies for concurrent engineering

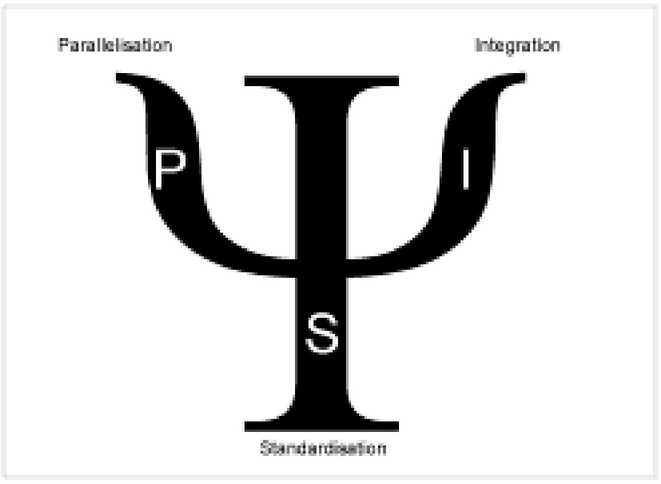


Figure 17: Strategies for concurrent engineering (PSI-strategy) (Berndes 1996)

Generally, three possible strategies can be identified as CE guiding principles (Figure 17):

- Parallelisation.
- Standardisation.
- Integration.

1. Parallelisation

Parallelisation in the product development process implies the cutting and optimisation of time. The first step is to remove existing float time in the development process. This means that processes, which do not have any dependencies on other processes, are carried out simultaneously. In practice most processes depend on others. In this case the dependent process has to be started before the preceding process is completed. An earlier start of the succeeding process is possible in most cases, because it can be carried out without having completed the preceding process. Not all information is required to start a new process. The result of this approach is the advantage of an accelerated execution of linked processes, but also the disadvantage of a higher decision complexity. This additional complexity is caused by an increased amount of information transfer between departments or teams. The proportion of uncertain and incomplete information is also higher due to the fact that not all parallel processes are finished which give inputs to other processes when they are started. In contrast to the Tayloristic principle, not only time can be cut due to parallelisation, but also amendment costs because of a lower number of mistakes. An example of parallelisation is the synchronisation of the development of product and means of production. The approach of parallelisation should be carried out under the principle that parallelisation does not mean to work side by side only but to work with one another.

2. Standardisation

Standardisation is defined as the unity of aspects in the product development process which show a high degree of similarity or the possibility of repetition. This is achieved by means of two basic approaches:

- *Structuring of processes*. Processes, which are often repeated, are specified and generalised.
- *Structuring of product*. This is the standardisation of products inclusive of its systems, elements and construction kit.

Standardisation is related to:

- Technical and structural aspects such as the usage of modules or components in the final product such as standard parts.
- Procedural aspects, structuring of operations and the definition of sequences of activities.
- Software standards such as ISO STEP ®, IAI IFC (Industry Foundation Classes).
- Aspects relating to the organisation of the structure such as interface between projects and departments. A clear definition of the organisation, i.e. standardised structures, is required to reduce and control the increased outlay of providing required information due to the implementation of CE.

The objectives of standardisation are to avoid repetition and needless work as well as to learn from existing experience of the company, industry or nation. Project staff can take repetitive and similar decisions quickly. Better co-ordination will be achieved. If routine tasks are optimised then theoretically more time will be available for innovative and creative work and for the management of unpredictable events. Standardisation should only be carried out if it is really necessary for parallelisation and integration (Berndes *et al.* 1996). Too much standardisation can lead to increased bureaucracy. Standardisation can vary from guidelines to compulsory arrangements and rules to fixed detailed operations.

3. Integration

If the product development process is seen as a uniform value-added chain then several departments such as R&D, sales, marketing, production and service are involved in the development of the product. The allocation of development tasks in different functional areas increases interface problems that result in the loss of information. The reason for the information loss is non-synchronised time scales, different interpretation of tasks and ignorance of the requirements at the other side of the interface.

Integration requires working in interdisciplinary teams and thinking and behaving in a process oriented way. There has to be the realisation that there is one common objective instead of several objectives that are department specific. The various departmental staff (or professionals in a construction team) must establish a view of the whole process that enables them to take appropriate action within their specific domains. Another important aspect of integration is data integration. A large proportion of construction data on large projects such as Menlyn is still in paper format. Integration will be greatly increased if more electronic information can be made reliable enough as to be trusted.

3.3.3.2 Concurrent engineering enabling technologies

The successful implementation of CE requires a convenient-to-use information technology infrastructure. To achieve parallelisation, standardisation and integration and to introduce and support throughout the entire product life cycle process the CE platform consists of three major components (Kessler 1996:104):

- A framework to model, support, control and integrate processes and teams. All the data necessary for the product must be produced within these processes.
- An information Management System to manage, change, release and store metadata related to the product.
- A Products Information archive to store and access a common product data model.

Abovementioned requirements were identified in the ESPRIT project CONSENS (Concurrent Simultaneous Engineering System) in 1992. The objective of this project was to develop an organisational and information technology concept to realise Concurrent Simultaneous Engineering in European companies.

Typical features built into abovementioned to support parallelisation are:

- Controlled and concurrent access to distributed data and information.
- Multi project management.
- Client-server architecture and multiple desktops.
- Interactions between different software packages.
- Modelling of independent and dependent processes to enable parallel and simultaneous work.
- Flexible reaction to changes in the product development process as well as in the organisation of the project.
- Support the user in adapting the installed project according to new requirements.
- Distributed database.
- Common method for executing tasks via the user interface.
- Visualisation of information and data flows to and from other processes.
- Structuring of the project or product into distinct interrelated or independent work packages, which can be worked in parallel.
- Possibility to divide the project into its components and flexible management in work packages.
- To support teamwork and parallel access of team members to different tasks of an installed project.
- To provide mechanisms to free information or data in time for other users to enable concurrent and simultaneous work in this project.

Standardisation:

- Software is implemented on different hardware platforms in a heterogeneous network.
- Distributed databases.
- Standardised interfaces to exchange data between different software tools and frameworks.
- Support of standardisation with the possibility to reuse results in multiple projects and to ensure their consistency.
- To allow the reuse of results and work packages.
- To prepare libraries for the reuse of processes and project tasks.
- To support the installation and reuse of standardised processes and projects.
- To provide functionality to model processes and their data interdependence.
- Reuse of existing components by an interface connected to external documents handling systems and archives.

Integration:

- To integrate different kinds of users such as supplier, designer and project manager and to provide each user with his customised profile of the integration platform.
- To provide a common graphical user interface for each user of the system.
- To allow the integration of domain neutral tools to support the user in controlling the processes.
- To offer interfaces for communication and integration.
- Access to different tools through the user interface of the integration platform.
- To run the integration platform as a distributed system to support different user locations.
- To use standardised interfaces and mechanisms to allow the exchange of data between different design tools.

- Use of standards for communication and integration.
- To manage the status of results and keep track of their consistency so that completely specified information can be distinguished from partially specified information.
- To manage all interdependencies between work packages and to inform the participants of the effects of their task.
- To provide the possibility to exchange information in a controlled and defined way between different processes.
- To execute tools necessary to fulfil tasks in a convenient and controlled way.
- To check the data transfer between processes.
- Multiple schemas, interrelated data and a shared data model.
- Object-oriented structuring of the real world.
- Consistency control and storage in the data-handling component provides a defined status of the data.
- To support a project or process oriented organisation and changes in the organisation of the current project.

3.3.3.3 Flow management

The storing of architectural design knowledge in the form of cases or a smaller more portable format will be in the form of an encapsulated environment. This environment when it is brought into the specific design project will form part of a life cycle process. At this point it will be governed by the flow patterns of the specific process environment. For this reason the author is of the opinion that a study of the basic flows in the construction development process phases gives an idea of the information flows, but fails to clearly identify production capacity of the user types. This has the effect that the criticality of various decisions and the lead times required to ensure proper synchronisation in the fast track or concurrent engineering project cannot be planned for.

Activities used in a design process cannot be invoked in an arbitrary order as data and time dependencies of activities have to be taken account of. To enable the user the possibility to define a set of activities in a specific order, flows are introduced. A flow defines time and data interdependencies between activities. Specific features of CAD and CASE (in a software engineering sense) that are used in a design or software engineering process cannot be invoked in an arbitrary order. The output data of one activity might be required as input for another. Figure 18 illustrates the basic different types of flows that are possible in any process. The rectangular elements represent activities or processes. Their interrelations are symbolised by an arrow, which presupposes that the activity on the left of the arrow has to be executed before the one on the right.

Both in the ESPRIT SCENIC project and local studies undertaken by Allen at the CSIR in 1999 information flows between the different users of project information in construction were extensively studied. The SCENIC project identified the following generic stages in the building life cycle:

1. Inception.
2. Briefing.
3. Feasibility.
4. Concept design.
5. Scheme or outline design.
6. Detail design.
7. Tender documentation.
8. Estimating and tendering.
9. Evaluation of tenders.
10. Off-site fabrication or prefabrication.
11. Delivery or logistics.
12. Production or assembly.
13. Testing, commissioning and hand-over.
14. Operation and facilities management.
15. Re-use or demolition (disassembly).

The author noted that in all cases the links between the different users, the time of occurrence and the nature of communication were recorded. However there are two major omissions. *Firstly* the diagrams fail to identify the throughput capability of the various different constituent processes. No research has been done on the time taken to achieve certain design and decision taking activities. This analysis is critically important to implement a successful CE system within the construction industry. The *second* omission is the fact that certain activities or even processes should be grouped together to ensure efficiency and modularity. This has a direct influence on the future sustainability and maintainability of systems.

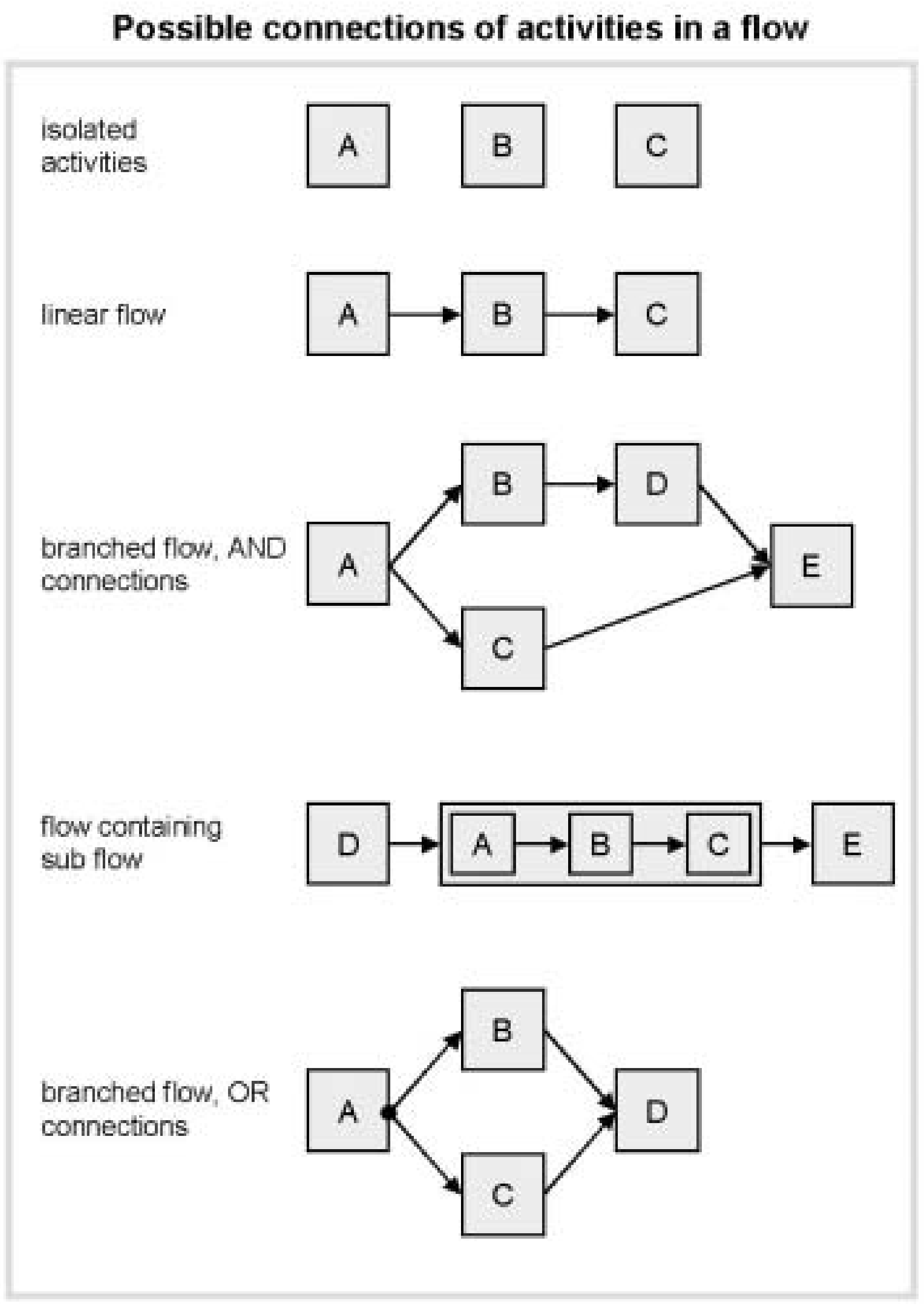

Possible connections of activities in a flow
isolated activities
A
B
C
linear flow
A
B
C
branched flow, AND connections
A
B
C
D
E
flow containing sub flow
D
A
B
C
E
branched flow, OR connections
A
B
C
D

Figure 18: Different flow types in a process (Author)

To the information in Figure 18 throughput information should be added (Figure 19). It is clear from a Voice of Customer (VOC) exercise, recently undertaken by the author that numerous bottlenecks can be identified during the construction process. Goldratt (1993:207) explained the impact on throughput in a process where bottleneck and non-bottleneck activities, equipment or even team members are combined (Figure 19).

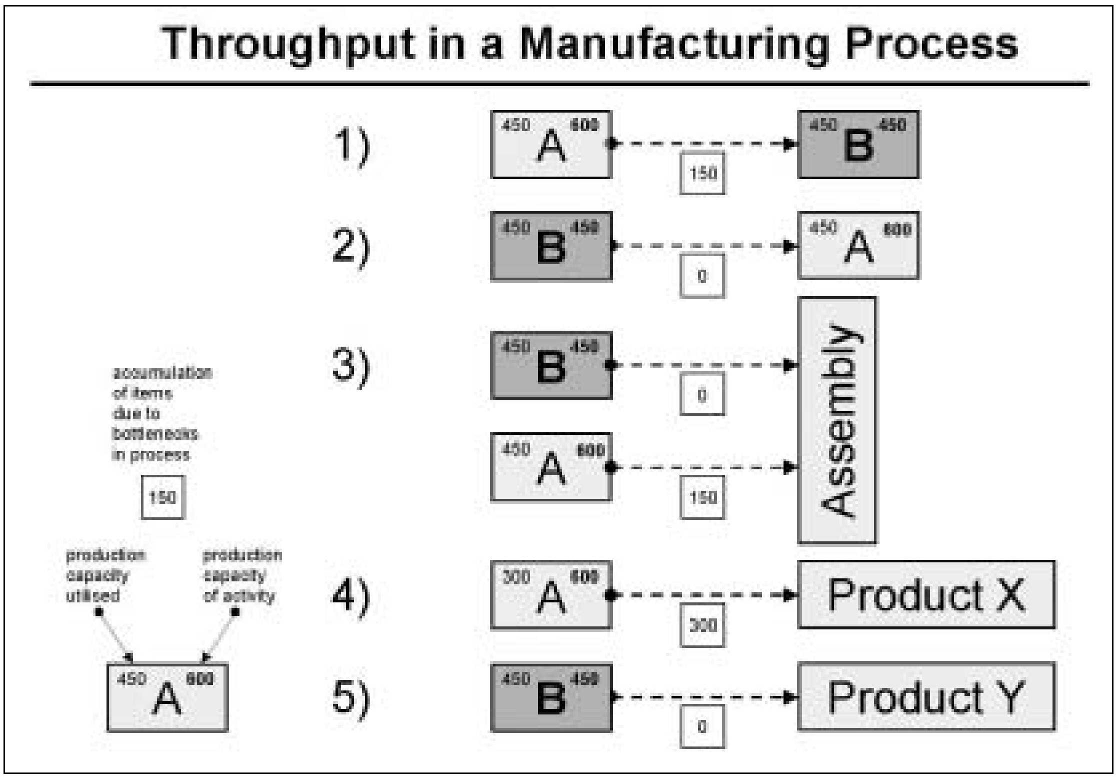

Throughput in a Manufacturing Process
1)
2)
3)
4)
5)
450 A 600
450 B 450
150
0
300
300 A 600
Assembly
Product X
Product Y
accumulation of items due to bottlenecks in process
production capacity utilised
production capacity of activity

Figure 19: Throughput in a manufacturing process (Author, based on Goldratt 1993:207)

In Figure 19, non-bottleneck activities (machines or workers) are designated with *A* and bottleneck ones with a *B*. The activities are connected by information or material flows. In each rectangular block the capability of the activity is indicated at the top right in bold numerals. The amount of the capability that can be utilised in each case is indicated by the value at the top left. The information flow consists of the following typical abstract and tangible activities and entities in the construction industry:

- Analyses (strategic, client and facilities analysis)
- Communication (appoints, reports, request, approves, program, brief, schedule)
- Design information (concept, scheme and detail design drawings, models)
- Construction material (pre-assembled and raw)
- Waste removal

Consider the various throughput types detailed in Figure 19.

Throughput type 1:

Non-bottleneck activity *A* is feeding bottleneck activity *B*. If *B* runs at full capacity, then 150 units will end up as inventory. In context this would mean unprocessed entities that cannot be handled by *B*. The production throughput of *A* is therefore effectively limited to 450 units.

Throughput type 2:

Bottleneck activity *B* is feeding non-bottleneck activity *A*. In this case, even if *B* runs at its full capacity, *A* will be starved of input, or it cannot run at full capacity. In this case only 450 units out of a possible 600 can be utilised. From this it can be concluded that the level of utilisation of a non-bottleneck is not determined by its own potential, but by some other constraint in the system.

Throughput type 3:

In this case output coming from both bottleneck and non-bottleneck processes are combined into a final product by means of assembly. In this simplified case, it is assumed that the final assembled product requires one item from *A* and one item from *B*. If process *A* runs at full capacity the effect will be that inventory (parts) will be manufactured that cannot be assembled into a final product, because too few parts are coming from the bottleneck *B* process. This has the effect that a significant amount of capital could be caught up in excess inventory. Goldratt (1993) identified the unexpected fact that a system running at full capacity is not necessarily an efficient system.

Throughput type 3 and 4:

In these cases there is no constraint in the system. Both *A* and *B* can produce at full capacity. However the constraint has now shifted from the internal processes to the ability to sell the products produced, in this case *X* and *Y*. Product X can only be sold at only 300 units per time unit. 300 units will therefore end up in inventory. In the case of product Y the sales force is able to sell all units per time unit. If the manufactured products cannot be sold, then capital will be tied up in inventory. If the sales tempo cannot be balanced with throughput the process could also become very inefficient.

3.3.3.4 Theory of Constraints (TOC)

Theories are usually classified as either *descriptive* or *prescriptive*. *Descriptive* theories, such as the law of gravity, tell us why things happen, but they do not help us to do anything about them. Prescriptive theories both explain why and offer guidance on what to do. TOC is in essence a prescriptive theory. Goldratt states that several principles converge that make the manufacturing environment particularly applicable for TOC. Goldratt identified the following TOC principles:

- Systems thinking is preferable to analytical thinking in managing change and solving problems.
- An optimal system solution deteriorates over time as the system's environment changes. A process of ongoing improvement is required to update and maintain the effectiveness of a solution.
- If a system is performing as well as it can, not more than one of its component parts will be. If all parts are performing as well as they can, the system as a whole may not be optimal. The system optimum is not the sum of the local optima.
- Systems are analogous to chains. Each system has a weakest link (constraint) that ultimately limits the success of the entire system.
- The strengthening of any link in a chain other than the weakest link (constraint) does nothing to improve the strength of the whole chain.
- Knowing what to change requires a thorough understanding of the system's current reality, its goal and the magnitude and direction of the difference between the two.
- Most of the undesirable effects within a system are caused by a few core problems.
- Core problems are almost never superficially apparent. They manifest themselves through a number of undesirable effects (UDEs) linked by a network of cause and effect.
- Elimination of individual UDEs gives a false sense of security while ignoring the underlying core problem. Solutions that do this are likely to be short-lived. Solution of a core problem simultaneously eliminates all resulting UDEs.
- Core problems are usually perpetuated by a hidden or underlying conflict. Solution of core problems requires challenging the assumptions underlying the conflict and invalidating at least one.

- System constraints can be either physical or policy. Physical constraints are relatively easy to identify and simple to eliminate. Policy constraints are usually more difficult to identify and eliminate, but removing them normally results in a larger degree of system improvement than the elimination of a physical constraint.
- Inertia is the worst enemy of a process of ongoing improvement. Solutions tend to assume a mass of their own that resists further change.
- Ideas are not solutions.

Goldratt (1993) states that to be productive you must have accomplished something in terms of the goal. Productivity is meaningless unless you know what your goal is. If the goal is to make money, an action that moves the company towards making money is productive. The high level measurements that are normally used to measure company performance is:

- Net profit
- Return on investment (ROI)
- Cash flow

The primary goal of any company is therefore to make money by increasing net profit, while simultaneously increasing return on investment and simultaneously increasing cash flow. In practical terms this can be stated as in terms of the operational rules *throughput*, *inventory* and *operational expense*. *Throughput* (T) is the rate at which the system generates money through sales. It is specifically sales and not production, because if you produce something but do not sell it, it is not throughput. *Inventory* (I) is all the money that the system has invested in purchasing things, which it intends to sell. *Operational Expense* (OE) is all the money the systems spends in order to turn inventory into throughput. Everything that goes into a process is covered by the relationship between these three operational measurements.

In order to improve a system the question is where the attention should be focussed. The theoretical limit in reducing OE and I is zero. A system cannot produce output with no *Inventory* and no *Operating Expense* and they are therefore somewhat above zero. Theoretically there is no upper limit to how high you can increase T, but as is apparent from Figure 19, there is a practical limit to the size of your market. The potential for increasing T is likely to be much higher than the potential for decreasing OE and I. It makes sense to expend as much effort as possible on activities that tend to increase T primarily and make reduction of I and OE a secondary priority (Dettmer 1997:17).

3.3.4 Taguchi techniques for quality engineering

According to Ross (1988) Taguchi addresses quality in two main areas namely off-line and on-line quality control (QC). Both off these areas are very cost sensitive in the decisions that are made with respect to the activities in each. Off-line QC refers to the improvement of quality in the product and process development stages. On-line QC refers to the monitoring of current manufacturing processes to verify the quality levels produced. Off-line QC is of particular importance in this study due to the fact that it is proposed that CBR/ CBD methods will be used in the design process. It is also important to improve quality as early as possible in the product life cycle. Taguchi methods should be seen in context with the other important methods discussed such as QFD and Kansei engineering.

3.3.4.1 The meaning of quality

Products have characteristics that describe their performance relative to customer requirements. Characteristics such as energy use of a house with regards heating of water, fuel economy of a vehicle and the strength of a door knob are all examples of products characteristics that are important to customers at one time or another. The quality of a product is measured in terms of those characteristics. Quality is related to the loss to society caused by a product during its life cycle. A high quality product will have minimal loss to society as it goes through this life cycle. The loss that a customer sustains can take many forms. It is generally a loss of product function or properties. Other losses are time, pollution and noise. If a product does not perform as expected the customer experience some loss. After a product is shipped, a decision point is reached. It is the point at which the producer can do nothing more to the product. Before shipment the producer can use expensive or inexpensive materials, use an expensive or inexpensive process, but once shipped, the commitment is made for a certain product expense during the remainder of its life cycle. This is of particular importance in the construction industry where the correct choice of lighting in a large shopping complex can save hundreds of thousands of Rands during the operational life of the complex.

Quality has but one true evaluator, the customer. The birth of a product is when a designer takes information from the customer to define what the customer wants, needs and expects from a particular product. Sometimes a new idea creates its own market, but once a competitor can duplicate the product, the technological advantage is lost.

3.3.4.2 Taguchi loss function

The Taguchi loss function recognises the customer's desire to have products that are more consistent and the producers desire to make a low-cost product. The loss to society is composed of the costs incurred in the production process as well as the costs encountered during use by the customer such as repair and lost business. A supplier in Japan made a polyethylene film with a nominal thickness of 0,991 mm that is used for greenhouse coverings (Figure 20). The customers want the film to be thick enough to resist wind damage but not too thick to prevent the transmission of light. The producers want the film to be thinner to be able to produce more area of the material at the same cost. At the time the national specifications for film thickness stated that the film should be 0,991 mm ± 0,203 mm. A manufacturer that made this film could control film thickness to 0,02 mm consistently. The company made an economic decision to reduce the nominal thickness to 0,813 mm and with their ability to produce film within 0,02 mm of the nominal the product would meet the national specification. The intention of this was reduce manufacturing costs and increased profits.

Unfortunately at the time strong typhoon winds caused a large number of the greenhouses to be destroyed. The cost to replace the film had to be paid by the customer. These costs were much higher than expected. What the producer had not considered was the fact that the customer's cost would rise while the producer's cost was falling. The loss function, loss to society, is the upper curve. This is the sum of the producer and customer's curves. This curve shows the proper thickness for the film to minimise loss to society. This is where the nominal value of 0,991 mm is located.

It is clear from the function that as the film gets thicker from the nominal 0,991 mm the producer is loosing money. On the other hand when the film gets thinner the customer is loosing money. The producer should fabricate film with a nominal thickness of 0,991 mm and reduce variation to that thickness to a low amount. If the manufacturer does not attempt to hold the nominal thickness at 0,991 mm and causes additional loss to society, then it is worse than stealing from the customer. If someone steals R10-00, the net loss to society is zero. Someone has a R10-00 loss and the thief gained R10-00. If the manufacturer causes an additional loss to society, everyone in society has suffered some loss. A producer who saves less money than the customer spends on repairs has done something worse than stealing from the customer. Subsequent to this experience the national specification was changed to make the average thickness produced 0,991 mm. The tolerance was left unchanged at ± 0,02 mm.

The cost of damages to the environment through energy production (externalities) are also a loss to society.

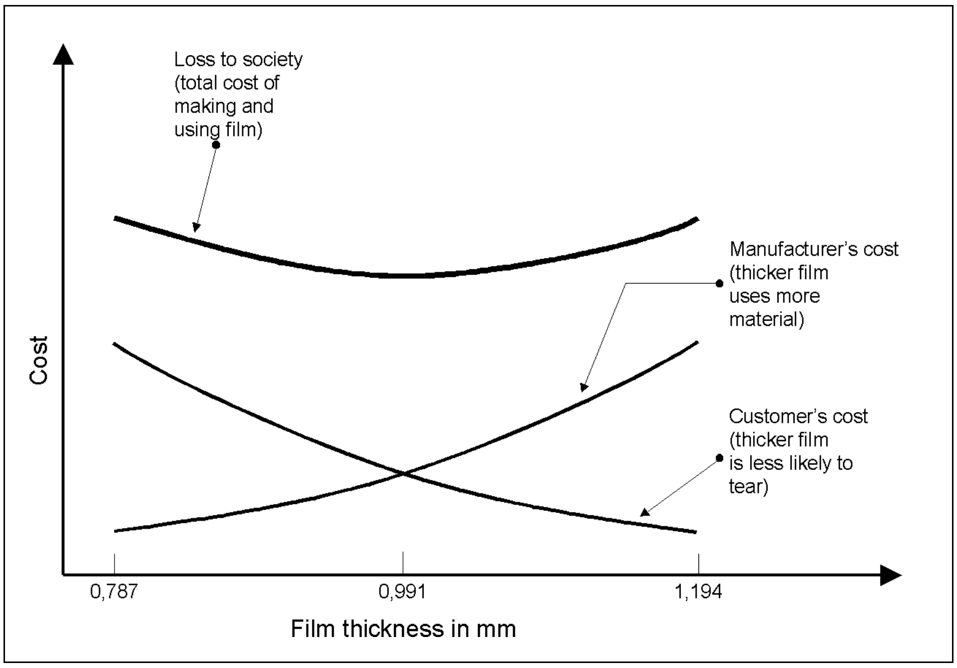


Figure 20: Costs associated with greenhouse film (Ross 1988)

3.3.5 The Fuzzy Front End (FFE)

In the high pressure environment of fast track (concurrent projects) time is an irreplaceable resource. The construction team should find opportunities to buy cycle time for less than cost. These opportunities appear throughout the development process. One place that is not often exploited seriously is the fuzzy zone between when a project opportunity is known and when we mount a serious effort on the development project. This approach is very different from conventional approaches that try to get a perfect solution at this stage by adding numerous checks and balances. The conventional logic is sound when markets are predictable and the cost of delay is low. However it breaks down in fast moving markets and when the cost of delay is high. This situation has been observed on large construction projects in this country.

Three critical factors combine to make the Fuzzy Front End an area of opportunity (Smith *et al.*, 1998):

- It lasts a long time.
- It is a cheap place to look for cycle time.
- Individual companies have big performance differences.

1. It lasts a long time.

Various delays occur right at the start of a construction project. There is a lot of time between when the team knows about a project and the time when a full development team started working on it.

2. It is a cheap place to look for cycle time.

If the typical actions that can be taken to buy a week of cycle time at various stages of the development process are analysed, enormous differences in cost are discovered. One consumer company spent $ 750 000 to buy three weeks of cycle time near the end of its development process by accelerating the shipment of critical capital equipment. This was a sound business decision, because the cost of delay on the programme was much higher than $ 250 000 per week. Yet the same three weeks could have been purchased for less than $100 a week during the FFE. This is 2 500 times cheaper!

3. Individual companies have big performance differences.

Some large companies plan so well that compelling market opportunities are lost due to the long period of time it takes to produce products. It has often been noticed that dynamic small companies can design and produce products long before the large company can even start. Instances have been noted where a small start-up company was 500 times faster than a large Fortune 500 company where well-intentioned planning and budgeting processes guaranteed defeat (Preston *et al*. 1998:52).

The following actions can be taken to improve the front-end processes:

- Institute metrics.
- Calculate the cost of delay.
- Assign responsibilities.
- Assign resources and deadlines.
- Capture opportunities frequently and early.
- Subdivide the planning.
- Create technology and marketing infrastructure.
- Create a strategy and a master plan.
- Prevent overloads.
- Create a quick-reaction plan.

Wheelwright *et al*. (1992:93) identified the primary types of development projects as:

- Enhancements, hybrids and derivatives.
- Next generation or platform.
- Radical breakthroughs.
- Research and advanced development.
- Alliance or partnered projects.

It is interesting to note the similarities between the construction industry and the *platform* development projects with regards the FFE. Platform projects represent the bundling and packaging of a set of improvements (design requirements) into a new system solution (design synthesis) for a much broader range of customer needs than the category of derivatives. Much creativity, insight and initiative are required at the FFE of a platform project than a derivative project.

3.4 Objects

3.4.1 Introduction

This section is included due to the dominance of objects in software engineering, CAD and interoperability. If the final application can be based on a generic platform using these technologies the likelihood of success is much higher. The author worked with one of the first object based CAD systems in the world i.e. GDS from Applied Research in Cambridge. The origins of this pioneering system through OXSYS, BDS and eventually GDS is described by Eastman (1999:53-61).

The concept of objects in software engineering, CAD and interoperability is dominating current software applications and new solutions proposed. Most modern programming languages claim to be based on objects. CAD systems such as MicroGDS, AutoCAD and MicroStation also claim to use object-oriented technology. To package software routines and data in the form of objects is very useful mainly because it keeps relevant, related data together. However objects are not the ultimate solution to all the problems studied in this thesis. This is mainly due to the fact that architectural design knowledge is generated at both tacit and explicit levels. The very fact that objects are encapsulated instances of a class implies that some higher order of system integration is required. This chapter explores various theories surrounding this very broad subject. The chapter is concluded with an analysis of the main industry standard object technologies available. In order to be successful in the complex knowledge driven environment we are currently operating in, it is very important that objects conform to certain essential requirements such as interoperability, www enabling and platform independence.

3.4.2 Origins of the object approach

The central concept in the object approach is that of the object. An object associates data and processes in a single entity, leaving only the interface visible from the outside. The interface gives the user access to the operations that can be performed on the object. This approach is not new, it appeared in the language Simula. Simula was designed as a structured programming language for simulating parallel processes. The classes of Simula made abstraction possible by hiding the implementation and creating increasingly complex entities.
The abstract aspect of the object approach, which enables a data structure to be hidden by the allowable operations for that structure was, formalised in the 1970s in the theory of abstract data types.

In parallel with this formalisation of abstract types, the language Smalltalk was developed. This language also implemented the object concept in the form of classes but added message passing taken from the actor concept and the use of inheritance to structure the classes hierarchically in terms of generalisation.

After Smalltalk the relations between generalisation and inheritance were developed extensively in artificial intelligence (AI) in connection with knowledge representation and more particularly in the context of frames and semantic networks.

Three points of view led to the object concept:

- Structural – the object is seen as an instance of a data type, characterised by a structure that is hidden by the permitted operations.
- Conceptual – the object corresponds to a concept of the real world, which can be specialised.
- Actor – the object is an active, autonomous entity that can respond to messages.

3.4.2 Why is the use of objects advisable

The object approach is characterised by the structuring of problems into object classes. The domains where this approach is used all require complex software that can handle large volumes of information. In the hope of controlling this complexity a number of objectives have been defined:

- Representation of real world entities without distorting or decomposing them.
- Re-use or extension of existing software.
- Development of environments rich in facilities such as tools for creating interfaces, debugging and for tracing execution paths.
- Rapid construction of high-quality graphical interactive man-machine interfaces, able to react to any external event such as a change of data.
- Facilitate the rapid prototyping of applications particularly for man-machine interfaces and general processing logic, without incurring the need for complete recoding.
- Facilities for exploiting parallelism when the software is implemented on multiple or distributed processor systems.

Objects can help the developer achieve these aims by their powers of abstraction, generalisation and interaction.

The object approach entails firstly defining the features of the objects that constitute an application and then making these objects interact by message passing. An object has a static aspect, which represents its state by means of instance variables or attributes. This is hidden by its dynamic aspect, which represents its behaviour and corresponds to the operations that can be performed on the object.

Object-oriented design is without doubt the main field in which the use of objects facilitates the work of the designer. It enables entities of the real world and the relationships between these to be represented directly. According to Meyer (1988) Object-oriented design is the method which leads to software architectures based on the objects every system or subsystem manipulates rather than the function it is meant to ensure. Object-oriented design is also the construction of software systems as structured collections of abstract data type implementations (Meyer 1988).

Object-oriented programming is unlike traditional programming. Objects are seen as active entities that perform their actions in response to messages sent to them. Instead of a software system program structure that consists of data and functions or procedures, a program is organised into active entities composed of data structures hidden by functions. The same function name can be used to perform similar actions on different objects, which makes it possible to construct an abstract language with which essentially different objects can be acted on in a similar manner.

An object approach program consists of a set of objects that exchange messages with each other, triggering operations (triggers or methods depending on the environment) that cause the internal state of the object to change and results to be returned.

An object-oriented database or object database (ODB) differs from traditional databases, because the real world objects are represented identically in the database on disk and in the application program in memory. An object is said to be persistent if its lifetime is greater than that of the program that created it and in this case it exists in the database. In traditional relational database management system an object is often decomposed in order to be stored into different database tables. In the object databases the object memory image is written directly to the disk.

It is estimated that by 1996 at least 80% of software developers were using object approaches (Bouzeghoub *et al*. 1997). One of the reasons for this gain in popularity is the fact that the object approach does not necessitate having pure object languages or pure object Database Management Systems (DBMS).

The market for object-oriented tools is very varied:

- Design methodology tools (CASE).
- Application Development Environments (ADE), including fourth-generation languages. These are usually used for building client-server applications for relational databases, using predefined graphical objects and a proprietary object language.
- Object-oriented programming languages (OOL) either pure object such as Smalltalk or hybrid such as C++.
- Object-oriented Database Management Systems (ODBMS). Either pure object or extensions of relational systems to include objects often called object-relational DBMS.
- Object-oriented middleware based on object request brokers (ORB) for passing messages between objects such as Common Object Request Broker Architecture (CORBA).

The four fields in which the object approach seems particularly important are:

- Programming, using either object-oriented extensions of existing languages (such as C, Pascal, Visual Basic) or pure object languages (Smalltalk, Eiffel or Java).
- Databases, using extensions to existing relational systems such as *Ingres*, *Oracle*, *DB2*, *Informix* or new systems, often based on OOL such as *Gemstone*, *ObjectStore*, *Versant* and *O2*.
- Design methods based on a combination of object-oriented models and specific representations. Objects have influenced most traditional methods.
- Distributed systems where distributed objects collaborate. As a result of the activities of the OMG, the object approach is bringing about a unification of the middleware products. This makes it possible to assemble objects over a network or even the Internet. Microsoft is important with its propriety approach at the core of Object Linking and Embedding (OLE) which is very likely to become a *de facto* standard in the near future.

3.4.3 Object-oriented programming

3.4.3.1 Encapsulation

Structured programming languages such as *Pascal* and *ALGOL* were designed with the aim of improving the structure of complex programs. They relate the processing to the data structure. In this approach there are three parts to an application program:

- Data structure
- Operations
- Main program

This approach was suitable if the application, albeit large, did not have to evolve a lot. It reaches it limits when the data structures or the procedures have to be shared by different programs and the data structures change with time.

Object programming solves these problems by encapsulation of the data and the operations that manipulate them in objects. This is an application of the principle of abstraction. An object is only accessible by means of its external interface operations that are visible. Its implementation is hidden from the programs that manipulate the object and have no effect on the programs that use it. Encapsulation thus ensures mutual independence among programs, operations and data. The advantage is that different programs can share the same objects without the need for import and export procedures.

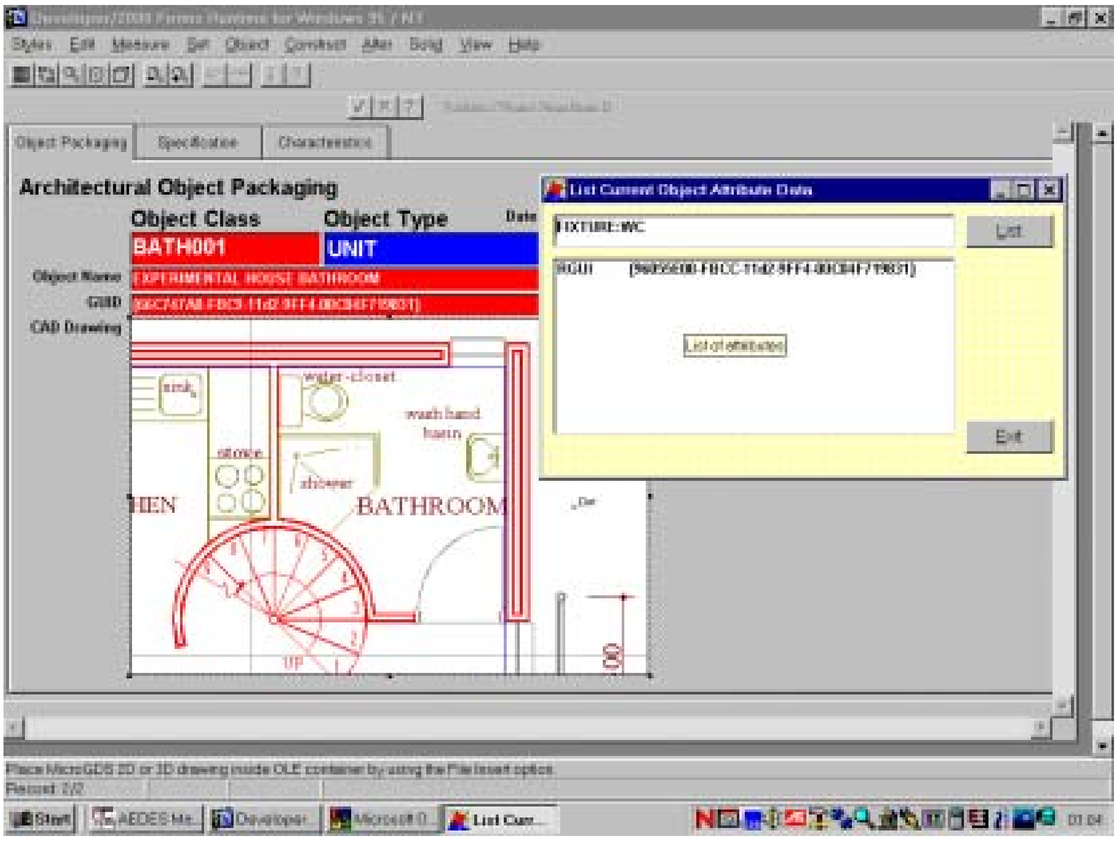
There are two very similar approaches that give a partial solution to the problems inherent in structured programming. The *modular approach* as used in the *Modula* language enables semantically related procedures to be grouped into modules that import or export procedures from and to other modules. This effectively encapsulates procedures in modules and thus makes it easier to represent the structure of an application. The *object-based approach* as used in the *Ada* language, extends the modular approach by adding abstract data types so as to make encapsulation of data structures possible. This increases the re-usability and extendibility of an application. Unlike the *object-oriented* approach the *object-based approach* lacks the concepts of inheritance and polymorphism.

3.4.3.2 Objects

Object and class are interdependent. An object is an instance of a class. A class is a logical grouping of objects having the same structure and the same behaviour. An object is an abstraction of a data item and consists of:

Object = identity + behaviour + state

An *object identifier* (OID) defines an object's identity. It is an unique and invariant attribute that enables the object to be referenced independently of all other objects. In the *AEDES* prototype system the Microsoft Global Unique Identifier (GUID) was used to this effect for the unique and persistent identification of CAD objects. The identifier is either generated by the system where the object is created or is packaged with the object during construction. An example of the former is the implicit linking type object names used in *PREMIS*. An example of the latter is the GUID pioneered by Microsoft to ensure global unique identification of *ActiveX* controls. In *AEDES* this was used to ensure unique identification of the graphical packaging of the starter kits (Figure 21).



Architectural Object Packaging

Object Class: BATH001
Object Type: UNIT
Object Name: EXPERIMENTAL HOUSE BATHROOM
GUID: {66C747A8-FBC9-11d2-9FF4-00C04F719831}
CAD Drawing

List Current Object Attribute Data

FIXTURE:WC

RGUI {96056E00-FBCC-11d2-9FF4-00C04F719831}

Place MicroGDS 2D or 3D drawing inside OLE container by using the File Insert option.
Record: 2/2

Figure 21: Using a Global Unique Identifier (GUID) to link graphic objects to other data (Author)

Figure 21 illustrates various different object concepts. The CAD drawing that contains the layout of a small bathroom has been inserted into an Oracle form. The particular data field of the form is defined as an OLE container. By double clicking on the CAD drawing the server for the CAD drawing is activated. The server is actually the CAD program itself, but the user is not really aware of it. Once the CAD server for the particular drawing is running it modifies the menu structure of the Oracle form to reflect the capabilities of the particular application running. In this case the command language of the CAD package uses Visual Basic as its command language. This small autonomous Visual Basic application with the *List* and *Exit* buttons can retrieve the attributes of a particular object within the CAD drawing. In this case it is responding back with the RGUI of the FIXTURE:WC graphic object. The Global Unique Identifier (RGUI) has been generated by the Microsoft utility *guidgen.exe* that is distributed with the Visual Basic language system. The R in RGUI indicates that we are dealing with a particular instance of the class of objects called FIXTURE:WC. In the particular CAD system an R as the first letter indicates that the attribute data applies only to this particular reference or instance of the object.

The state of an object is a value that can either be simple, a literal, or structured, for example a list. In the latter case it can be composed of simple values, referenced to other objects or values that are themselves structured.

The behaviour of an object is defined by a set of operations that can be applied to it (the methods) and are defined in the class to which the object belongs.

An object is an abstraction of a data item characterised by a unique and invariant identifier, a class to which it belongs and a state represented by a simple or a structured value.

Two objects O_1 and O_2 are identical if their OIDs are equal. They are equal if their states are equal. For objects O_1, O_2 we write $O_1 == O_2$ if O_1 and O_2 are identical and $O_1 = O_2$ if their states are equal. This implies, $O_1 == O_2 \Rightarrow O_1 = O_2$.

3.4.3.3 Class

Objects of the same nature, for example a CAD representation of a building component, will generally have the same structure and behaviour. The class expresses the common features and is a means of classification.

Class = instantiation + attributes + operations

A class provides the mechanism of instantiation that enables a new object (design object in *AEDES* terms) to be created. The object that is thus created is an instance of the class. The set of all instances of a given class is the class extent. In the *AEDES* prototype eight hierarchical classes of architectural objects were identified:

• Complex	(Hospital complex: Group of buildings)
• Facility	(Hospital building and site)
• Department	(Administration department)
• Unit	(Bathroom)
• Zone	(Wet area)
• Building Element	(Door)
• Component	(Door lock set)
• Sub-component	(Screw)

A class is also an abstract data type, which specifies the attribute and behaviour of operations for object instances of the class. The instance attributes have a name and a type. The operations are the operations that can be applied to an object belonging to the specific class.

A class is an abstract data type characterised by a set of properties (attributes and operations) common to its objects, with a means for creating objects with these properties.

The principle of encapsulation ensures that the attributes of a class can only be accessed externally by means of the operation (method) of reading that is provided.

3.4.4 The model approach to architectural design

In the past many attempts were made to structure architectural design in the form of full 3D models. One of the earliest attempts was *OXSYS*. All these attempts failed to achieve full automation of the design process or full quantification of the various design factors. Richens (1994) states that the Knowledge Based System (KBS) community is over-optimistic in their analysis of the nature of design. The KBS community (Carrara *et al.* 1994) typically characterises design as:

- Defining a set of functional objectives that ought to be achieved by the design artefact.
- Constructing design solutions which, in the opinion of the designer, are (or should) be capable of achieving the predetermined objectives.
- Verifying that these solutions are internally consistent and achieve the objectives.

However, the abovementioned is only a part of what really goes on in architectural design. Architectural objectives usually include functional ones, but are dominated by less definable intentions and are never collected completely before design starts. They evolve and are discovered as the work proceeds.

This is not due to architects that are badly trained, but more due the nature of the problem. Yet an approach that starts with a requirement that leads to derived functions and to design objects is useful if the intention is to package the explicit design knowledge. Pugh (1996) states that design is not only the integrative mechanism that brings together the arts and sciences but also the culture which envelopes both. Design is not like mathematics or physics. It does not represent a body of knowledge. It is the activity that integrates the bodies of knowledge present in the arts and sciences. The author is of the opinion that design is both an integration between arts and sciences in a horizontal dimension, but also an integration between tacit and explicit knowledge in a vertical dimension over time.

The reasons that could be identified why an electronic model approach to architecture is only partially successful and why attempts towards interoperability are likely to fail are the following:

- Model based approaches assume that all design knowledge must reside in a single model, or a set of interoperable objects. This approach assumes that all design knowledge can be quantified.
- It is assumed that working drawings such as plans, elevations, sections and details can be produced from the model. In *OXSYS*, *BDS* and other subsequent systems this failed because architectural drawings contain notations such as specifications, dimensions and general notation that can only be conveniently placed on 2D extractions of the 3D model. These extractions to 2D worked well, however if any changes are required then changes had to be made in the 3D model. This was time consuming and inefficient.
- The model approach captures only the functional manifestations of the design process, not the invisible tacit and experience aspects.
- Intelligence in a CAD system is inversely related to flexibility. The more detailed you wish to make the data model, the more circumscribed is the Universe and so flexibility is lost.
- Even after 5 years the International Alliance for Interoperability (IAI) failed to solve the problem of the intelligent internal interoperability of design objects such as a door in a wall. The IAI is likely to achieve extraneous interoperability due to standardisation of the design objects. Even at this stage the definitions of internal design objects are very incomplete and still require a significant effort to bring about a new industry standard.

3.4.5 Frameworks for object components

The use of objects improves technologies such as languages, tools, databases and other technologies that are essential for the construction of complex applications. The objects that can be produced by means of these technologies are very heterogeneous, particularly in size and level of abstraction. To assemble these into an application creates very difficult problems. The task can be simplified by using middleware products such as *CORBA* for communicating between these heterogeneous objects.

An *object component* is an independent, autonomous object, capable of co-operating with other objects in a distributed system with the intention to provide a globally available service to the application. Components were initially developed to ease the work of system developers and integrators where the synonymous terms *technical object* and *technical component* had their origins. More recently the concept of *business object* came into use.

Object middleware is a software bus that enables any object component to inter communicate in a manner that is transparent to the network and to the specific software application. It must be convenient to create, deploy and maintain complex systems. The components should be extendable and adaptable to specific needs and be capable of being combined dynamically in many different ways. Object frameworks offer an architecture into which the object components can be integrated and co-operate. An object framework provides tools for creating and assembling the components. The development of such frameworks, a very important requirement in the object world, is a great challenge. Only a few designers have mastered the technology of middleware. Recently frameworks for compound documents such as Object Linking and Embedding (OLE) developed by Microsoft appeared. The metaphor of a document is used for the integration of components.

3.4.5.1 Aims of object components

A component should have the following properties:
- Standard interface.
- Encapsulation
- Extendibility.

Standardisation of interfaces is the only way to ensure that independent components, developed with different programming languages, can co-operate. Encapsulation ensures that as components evolve they will not impact on the interface and the application that must use it.

The creation of abstract superclasses improves the extendibility of a software product. The superclass defines the general behaviour common to all possible classes. If a new special subclass is desired all you have to do is to implement a new subclass with only the specialised behaviour that is different.

The term object component includes many different software products such as libraries of classes, graphical interfaces, compound-document components and business objects. Three main categories can be identified:

- Technical components.
- Compound documents.
- Business components.

3.4.5.2 Technical components

These are libraries of classes developed by programming languages such as *C++* or *Smalltalk*. They are normally generic and extend the services provided by the language. In the *PREMIS* parametric symbol programming language, *Symbolix*, special components are provided to facilitate the programming of complex graphic visualisations. They are sometimes specialised for a particular activity such as access to relational databases or the development of graphical interfaces. The use of technical components is more along the classical lines in an *object-oriented* manner, by including classes from these components in the application program.

3.4.5.3 Compound documents

A compound document is an electronic document into which diverse components can be incorporated. Such a document is typically presented as a graphical window in which each component has its own graphical interface. The document metaphor is used extensively in the personal computer world. The document is generalised to enable objects of a variety of types to be edited and visualised in a uniform manner. In contrast to the cut and paste approach to joining heterogeneous objects, the compound document approach is essentially dynamic. This enables the object component to be manipulated directly from the document in which it is contained. This has the benefit of simplifying the creation and maintenance of complex documents.

Use of this approach requires many document components to be available in particular multimedia components. Technical components are easily integrated into a compound document.

3.4.5.4 Business components

A business object is an element of a typical business field of activity such as that of a bank or manufacturing company with rich semantic properties such as name, attributes, interfaces, relationships and constraints. Typical business objects for a manufacturing company may be address book, customer management, warehouse management, cash management, purchases, orders, finances, parts and deliveries. The objects must be able to communicate at a high semantic level. The company Discon uses a business object approach (Engelbrecht 1998). They use the technique of Functional Effect Back tracking to calculate the most appropriate grouping of these objects. The intention is that if a major component such as the financial module is replaced with a newer one it should not in any way destabilise the existing rest of the components.

Business components are of large size, with classes composed of many subclasses. They are unlike technical components and components of documents that are designed to facilitate the re-use of code. They are created in a top-down manner by means of object-oriented analysis and design.

Due to the availability of powerful CASE and 4GL tools more and more time is devoted to the analysis and design of business systems. Business components are becoming increasingly important in giving structure to applications. Enterprises are developing these components to meet their own needs for interoperability and re-use of applications.

The Business Object Model Special Interest Group (BOMSIG) of the OMG is working on the standardisation of business objects. It proposes to define these in terms of three types of object:

- Presentation, for managing the object’s graphical interface.
- Business, for managing the object on disk.
- Process, for realising complex operations on the business objects.

Objects of these three types can be heterogeneous and will communicate over a distributed architecture such as *CORBA*.

3.4.6 OLE/ COM from Microsoft

In 1994 Microsoft began to introduce Visual Basic custom controls. Today these controls are known as *ActiveX*. Microsoft is actively promoting the use of a new object-component model (COM) for the efficient management of distributed objects and the incorporation of these into compound documents. Today it is possible to use these controls in a wide variety of environments such as *Microsoft Word*, *Excel*, *Visual Basic*, *Access*, the www and other third party products such as *Arena* and *Visio*.

The problem of the fragile base class in classical models such as *C++* inhibits distribution of components and particularly successive versions in binary. Classical object models are therefore unsuitable for distribution over a network. This problem motivated the development of dynamic languages such as *Java*. Microsoft's aim of code exchange in binary implies the need to define a means for invoking objects at binary level, independent of any language. The *CORBA* approach, which specifies the interface in terms of a language, is not suitable.

Management of compound documents is an important objective of producers of distributed component frameworks. Microsoft is seriously committed to this route with its OLE architecture that is based on Component Object Model (COM).

There are two possible methods for integrating objects into a compound document, i.e. linking and embedding. The linking option uses a pointer in the forms of a name or identifier into the document. The linked object continues to reside in the original source document. The advantage of this is that it does not increase the size of the document and allows for multiple applications to share the object. In the case of embedding a copy of the original source object is inserted into the current document. This increases the size of the document but allows the linked object to evolve independently.

A framework for compound documents offers various basic services when components are assembled, stored and modified. The main services are persistence, saving objects, data exchange, construction of documents and interactive activation for making these active and modifying them.

3.4.6.1 Persistence of objects

Containers can be saved to disk and returned to memory when needed. It is important to remember the type of every object component. Microsoft organises the objects into compound files each containing a number of subfiles.

3.4.6.2 Data exchange

The facility of exchanging data between documents enables parts of documents to be copied directly into another. It is an essential requirement for drag-and-drop. This service can make use of the more rudimentary cut-and-paste service that uses the clipboard as an intermediary place of storage.

3.4.6.3 Enabling relationships between documents

A basic relationship service is necessary for enabling one document to point to another or to an object. In the case of simple compound documents the relationship is logical inclusion. The management of hyper documents requires other types of links to construct the hyperlinks. For compound documents aggregation is the only type of link that is handled. The invariance of the pointers creates special problems. Use of absolute file names with offsets inside the file is prohibited so as to ensure this invariance in the face of modifications. OLE uses names of files or subfiles, but these are modified if the reference is changed.

3.4.6.4 In-place activation

Activation occurs when the object is selected in the document window. The document remains under the control of its application or of an adapted application. This facilitates visualisation and editing. The editing menu of the main application is changed to reflect the needs of the object under consideration whilst maintaining full synchronisation.

In the case of OLE this is achieved by contacting the application that created the object under consideration. It continues to load it under a server process (if it is not already active) and transfers control to the server.

3.4.6.5 The object-component model

COM supports classes. A class is being implemented as a set of functions provided by servers in the form of executables (EXE) or by libraries (DLL). Encapsulation is total and data are hidden. Only the functions are visible. A class has a 32-bit global identifier (GUID) that is generated by an OLE utility or for basic classes specified by Microsoft.

An object is an instance of a class, with hidden data. In general it will have several interfaces each with an identifier and accessible at binary level by a pointer. Clients communicate with an interface by means of this pointer. This can be obtained through the medium of the interface identifier. It references a table of functions of the interface, given in the order they were specified.

To enable users to find the interfaces, each object is provided with a standard interface *IUnknown* through which its other interfaces can be found. The interface to *IUnknown* is thus a root from which all classes providing basic functions descend. It enables a user to point to any object and obtain at least a pointer to *IUnknown. IUnknown* also enables counts of references to be kept, so that the memory space of objects no longer being referenced can be reclaimed.

Inheritance of structure does not exist in COM. Inheritance of structure is replaced by objects with multiple interfaces. This enables modification of components without incurring the need for recompilation. It also makes versions of interfaces possible. The existence of one interface through which the others can be found means that objects are self-documenting.

3.4.6.6 Support for distributed objects

This enables access to the interface of distant objects, managed by a different process EXE. The server process can be local or distant. OLE ensures transparency of the type of server for all client objects. Different components are brought into play for this purpose and a service is provided for localising the server and initiating execution. This service, service control manager (SCM), localises the server by first consulting the system directory and then activating the server. If it is already active it will establish contact with it. Message passing is based on the RPC of DCE. It is a matter of creating an issuing proxy object in the client and a receiving stub in the server. This mode of dialogue takes place over a proprietary software bus.

3.4.6.7 OLE/ COM basic services

COM and OLE provide the services essential to distributed systems. The basic services are implemented more in COM and those relating to compound documents more on OLE. Some of the services provided are:

- Persistence, for storing and retrieving distributed persistent objects.
- Exchange, for transferring data between components in a uniform manner.
- Relationships, for implementing links by means of intelligent names.
- In-place activation, for assembling and activating multiple components in a container.
- Automation for dynamic invocation of typed objects from programming languages such as Visual Basic and Visual C++.

The functions essential to these services are described below. They are grouped according to the interfaces to the objects that use the services.

Persistence

This provides for saving and restoring collections of objects either directly or by means of links within a single file. The container receives the application objects in memory pages and its contents can be saved in a compound file. There is therefore a hierarchy of files consisting of data elements and directories held in a single physical file.

Transaction management is used. Writing is in transaction mode, with either complete validation at the end of the transaction (*commit*) or cancelling all the updates performed in the course of the transaction (*revert*). Files can be modified incrementally without the need for a complete rewrite. Files are shared, making it possible for data to be exchanged between processes. The basic interfaces provided by the file management service are as follows:

IStorage(Record->Create,Open,Copy,EnumElementTo, …)
IStream(File->Create,Read,Write,Seek, …)

Data exchange

COM provides a uniform data transfer service, for which there is a standard interface *IDataObject.* This interface provides functions for recording in memory and retrieving data in various formats. The basic functions are *GetData*, *SetData* and *EnumerateFormat.* Objects with the interface *IDataObject* can exchange data directly without needing to go through a clipboard as is required in Windows. The interface also enables a user to be advised of a change in the source.

On top of *IDataObject* the drag-and-drop service enables data to be moved from a source object to a target. This is by means of a pointer to an *IDataObject.* The service acts as a mediator between the source and the target. The two must have the interfaces *IDropSource* and *IDropTarget* respectively.

Relationship

This enables objects to be linked to or incorporated into compound documents. It manages relationships that are simple aggregations. A link references an object from a compound document. It is implemented by a persistent name, a *moniker*. A moniker is an object that implements a link. Monikers support composite names, relative names and a function *BindToObject* that gives access to an object.

In-place activation

An object that forms part of a compound document is activated by a double click. The application that created the document is loaded as a server and takes over control. It can act either in-place or in a new window. The document can be edited directly on-screen. Linked or incorporated objects can also be activated in place, without the need to create a new window. The application takes over the document window and the object becomes the active agent that controls the keyboard. Only incorporated objects can be modified. Several interfaces are needed for in-place editing. The container must support *IOleInPlaceSite* and *IOleInPlaceFrame* and the object must have been given *IOleInPlaceActiveObject.*

Automation is a key OLE service that enables the functions of an application to be described (EXE), incorporated into OLE and made dynamically callable by any language that can use scripts. A good example of this is the *VBScript* sub-set of Visual Basic that is now extensively used in Internet web pages. The interface is described in the *Object Description Language* (ODL). Object descriptions are held in a *type library*. The type libraries can be managed directly by the *ICreateType* interface for creation.

A client OLE automation controller invokes an OLE automation server. The invocation is dynamic and is passed by the interface *IDispatch*. *IDispatch* receives the function Invoke from the client, decodes its parameters and sets up a link with the server. It calls the required function and passes the parameters.

3.4.6.8 The main OLE interfaces

Microsoft introduced the OLE architecture in 1991 for the purpose of managing compound documents. Since 1994 all the interfaces described above have been combined in a single context. Multiprocesses are supported in Windows NT and single processes in Windows 95/ 98. Recent distributions of Visual Basic include the distributed version of COM, DCOM. This has the effect that OLE is steadily evolving from its original use in compound documents to a distributed system where any client can communicate with any server.

The kernel implements COM and brings together the basic functions of persistence of objects, management of intelligent names and uniform transfer of data between objects. OLE is built on top of COM or DCOM. It provides management of compound documents by means of in-place visual editing and drag-and-drop of objects between documents. It also supports nesting of linked or copied objects and management of relative links.

Microsoft provides a very complete architecture. It is object based rather than object-oriented. Unfortunately it does not support inheritance. The services and framework provided is very complex. The architecture goes well beyond the handling of compound documents. Distribution is achieved by means of message exchange between proxies and stubs using RPC.

Objects are multi-interfaced providing a kind of multiple inheritance with convenient properties. Microsoft is already using a component approach, based on OLE in many fields such as operating systems, databases and multimedia. Other suppliers can enhance Microsoft’s products by incorporating their own components.

3.5 Kansei engineering and new product development

3.5.1 Introduction

Kansei Engineering (KE) is one the lesser-known product development techniques. Due to the enormous influence that the Japanese had in the domain of product development this technique was included. Figure 2 indicates that KE is one the techniques that is able to extract tacit needs from the level of *unexpressed thought*. This is exactly the area in design where very little has been achieved in design systems over the past 35 years. KE is also one of the few proven techniques that can operate at this high tacit level. It is envisaged that some support for KE be included in the final product.

The term KE was first used in 1986 by Kenichi Yamamoto, the current chairman of Mazda Motors, in a special lecture given at the University of Michigan. Thirty years ago companies could easily make a profit because it was a seller's market. The product development strategies of the time were based on product output. With the increasing saturation of the market product developers and marketing had to pay increasingly more attention to quality to differentiate their products. Today consumers demand good quality products. The economic success of a manufacturing firm depends on their ability to identify the needs of customers and to quickly create products that meet these needs and can be produced at low cost (Ulrich *et al.* 1995:2). To achieve these goals is a marketing, design and manufacturing problem. The totality can be called product development or total design (Pugh 1996). The basic needs manifested itself in movements such as the Total Quality Movement (TQM). Moss (1995:4) states that TQM is both a philosophy and a set of guiding principles that represent the foundation of a continuously improving organisation. TQM is the application of quantitative and human resources to improve the material and service supplied to an organisation. It is also the degree to which the needs of the customer are met, now and in the future. KE is one of the methods that can be used to quantify the higher order tacit feelings of the consumer into a product.

Nagamachi (1999a) is of the opinion that even in a strong economy, like that of the U.S.A., one of the reasons why some products sell well and some not is due to fact that not all products focus on consumer feelings and emotions. He calls this Kansei. The use of KE could provide quantified knowledge that can potentially be encapsulated in object-oriented architectural design packaging (Figure 1). Zultner (1999:360) identified KE as one of the methods to define customer needs that concentrate on the emotional responses.

3.5.2 What is Kansei Engineering (KE)

It is a technology that attempts to quantify cognition and product image in such a way as to influence the product development process (Figure 22). Like Quality Function Deployment (QFD), Kansei is a technique that is consumer-oriented and attaches importance to the voice of the customer. Nagamachi started KE at Hiroshima University about 25 years ago. It is an ergonomic consumer-oriented technology for new product development. In Kansei engineering Nagamachi focuses on three main aspects:

- Accurately understanding of consumer Kansei.
- To translate the quantified Kansei values into the product design.
- To create a system and organisation for Kansei-oriented design.

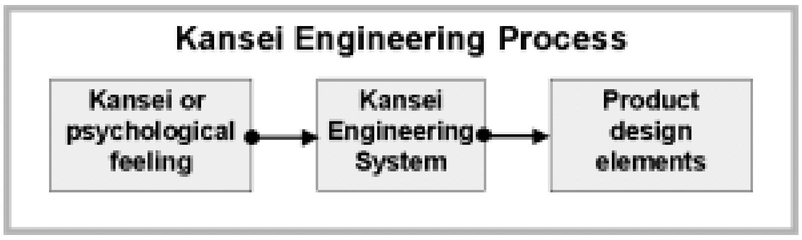

Kansei Engineering Process
Kansei or psychological feeling
Kansei Engineering System
Product design elements

Figure 22: The Kansei engineering process (Nagamachi 1999)

The word Kansei encompasses the following meanings:

- A feeling that one holds about a certain thing that may or may not exist but is thought to help enhance one's quality of life.
- All feelings and emotions that one has about a product, including its functions and appearance.
- Vague psychological emotions and senses that one holds but is not yet expressed.

The techniques used in Kansei attempts to quantify human *cognition* through the six senses of *vision, hearing, smell, taste, touch* and *inner sense (feeling).* For example you would walk into a building and approach the receptionist's desk. You rapidly form a first impression. You might feel that interior design is very modern *(vision),* the receptionist is very professional *(cognition)* and the acoustics is very good due to the soft carpets, wall and ceiling construction *(hearing).* Kansei initially sounds vague and ambiguous. However with special methods the feelings can be made tangible for the specific product where they manifest.

KE is a technology to translate Kansei into the design domain. Presently well-developed methodologies and software systems support it. The general input sources of KE data could be:

- Physiological data. When using physiological data to measure Kansei, the software can be used to determine data patterns.
- Psychological data can be handled by means of the sorting of data into clusters using a neural network model, classification of the data by means of genetic algorithms and breaking down the data into design elements using the quantification theory.

The latter method is most often used because of its convenience.

3.5.3 Types of Kansei Engineering

Five main technical types of KE is most often used:

- Type 1: Category Classification. It identifies the design elements of the product to be developed, translated from the consumer's feelings and image.
- Type 2: Kansei Engineering System (KES). A computer aided system, with an inference engine and Kansei databases, is used.
- Type 3: Hybrid Kansei Engineering System. The dual software systems of forward KES that goes from Kansei to the design specifications and reverse KES that goes from design specifications to Kansei is used in this type.
- Type 4: Virtual Kansei Engineering. This is an integration of virtual reality technology and Kansei engineering in a computer system.
- Type 5: Collaborative Kansei Engineering Designing. Group work design system utilising intelligent software and databases over the Internet.

3.5.3.1 Type 1: Category Classification

Category Classification is a method by which a Kansei category of a planned target is broken down into a tree structure to determine the physical design detail. Mazda used this type of KE for the new “Miata” (Eunos Roadster in Japan). KE became the fundamental technology for new product development at Mazda.

In the case of the “Miata” the project team decided that the zero level product purpose (mission or aim) would be “Human-Machine Unity” (Figure 23). The team broke the zero-level concepts into subconcepts level 1, 2 to n. In the case of the “Miata” the zero-level concepts were *tight feeling*, *direct feeling*, *speedy feeling* and *communication*. The various feelings are translated into physical traits, ergonomic specifications and automotive design elements.

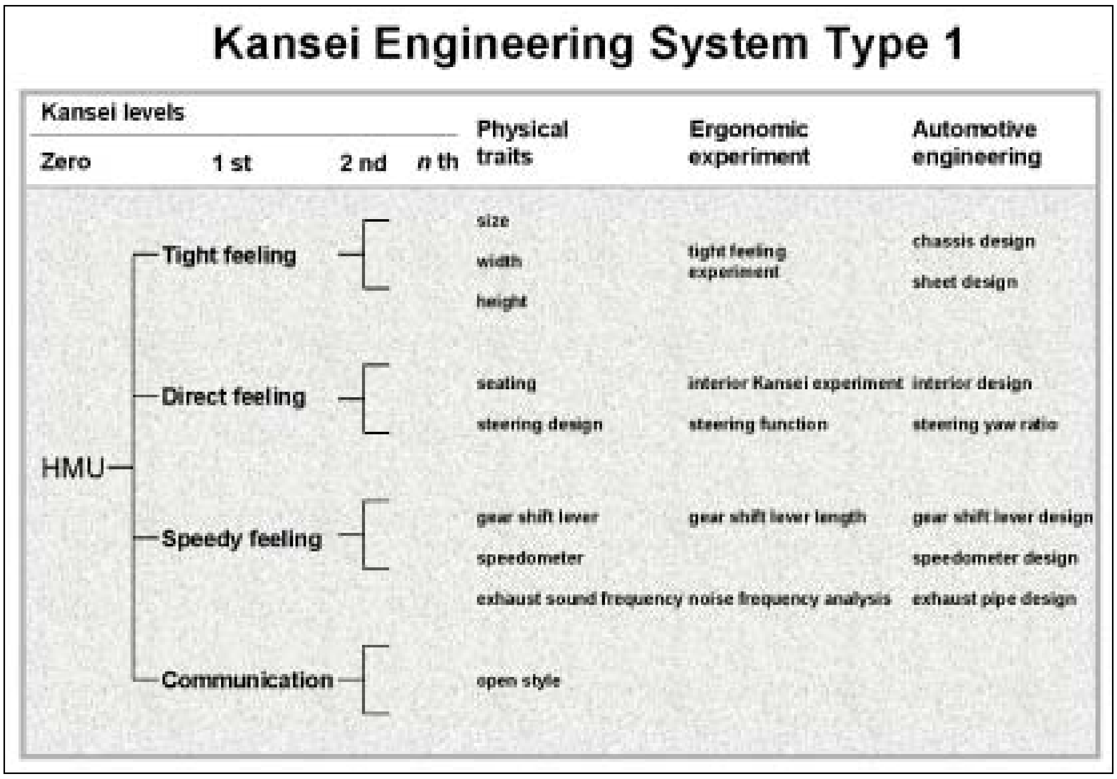


Figure 23: The translation of Kansei into physical car traits (Nagamachi 1999)

3.5.3.2 Type 2: Kansei Engineering Computer System (KES)

The KES is a computerised system with an Expert System that supports the transfer of the consumer’s feeling into physical design elements. The KES has four databases and an inference engine in the KES structure (Figure 24). The following databases are used:

- Kansei Database. Kansei words used in the new product domain are collected. Typically 600 to 800 words are initially collected. This is reduced to approximately 100 words that best describe the new product. These words are normally adjectives and sometimes nouns. In the automotive industry words such as “fast”, “easy to control” and “gorgeous” are used. After an ergonomic evaluation has been conducted these Kansei words are analysed by multivariate techniques such as factor and cluster analysis. The Kansei database contains the statistically analysed data. The KE is conducted by means of a Semantic Differential (SD) method on a five-point scale.

- Image Database. Data evaluated by SD scales are then further analysed using Hayashi's Quantification Theory Type II (Nagamachi 1999b). It is a multiple regression technique for qualitative data. The statistical relationships obtained between Kansei words and design elements constitute the image database. This database relates the most appropriate design elements to Kansei words and vice versa.

- Knowledge base. The knowledge base is a rule based database in *if then* form. It controls the image database. It also includes design guidelines and digital colour expression system.

- Shape and colour databases. The design detail is implemented in a shape design and colouring database. The parts are design aspects that are co-ordinated into the final assembled product with each Kansei word. The colour database consists of colours co-ordinated with Kansei words. The design and colour is extracted by a purpose made inference system based on the rule-base and is graphically displayed on the screen.

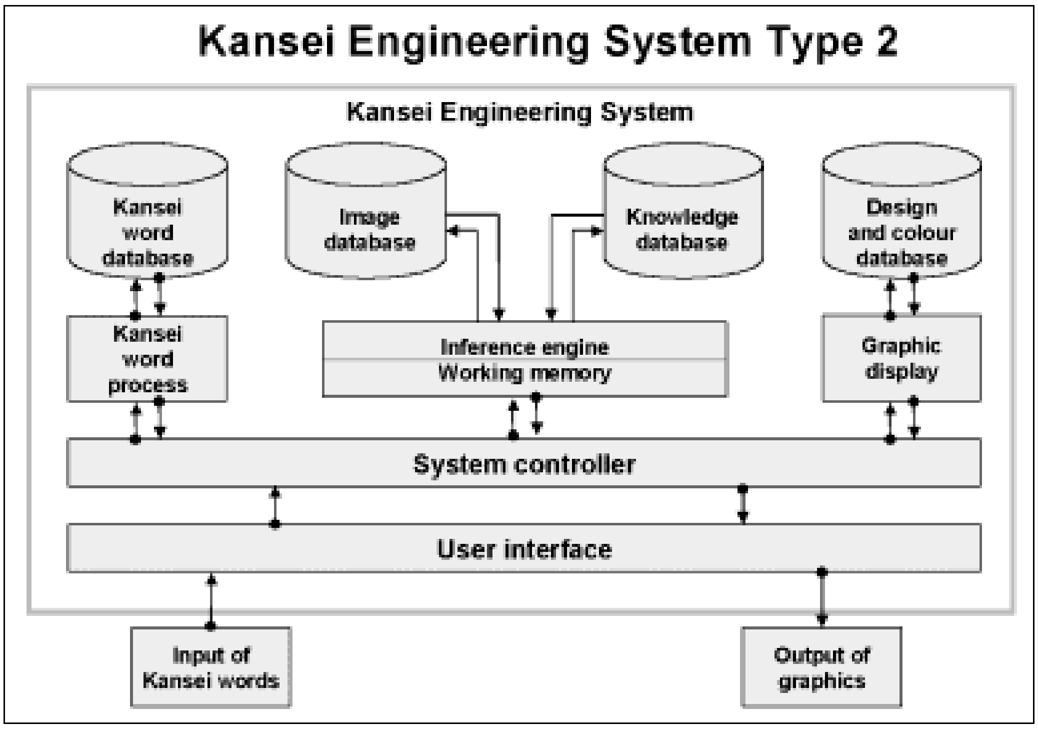


Figure 24: Type 2: Kansei Engineering Computer System (KES) (Nagamachi 1999)

3.5.3.3 Type 3: Kansei Engineering Modelling

KE type 3 uses a mathematical model constructed in the computerised system in stead of a rule-base system as described above at type 2. The mathematical model is based on Fuzzy Logic. Sanyo attempted to use Fuzzy Logic in an intelligent colour printer that could enhance bad original images. The developers carried out an experiment on an image of a beautiful girl and obtained data of the hue, brightness and saturation for a girl's face colour which are represented by a membership function in Fuzzy Set Theory. The data were transformed to Red: Green: Blue in the computer colour system. This enabled the KE colour printer to enhance the original picture by means of the KE inference system. The intelligent colour printing system comprised a camera, computer and a colour printing system driven by Fuzzy Logic. It was able to diagnose the original picture.

Nagamachi also developed a computerised language analysis system for the Japanese language to analyse words in terms of Fuzzy Integral and Fuzzy Measure Logic. It is used to analyse brand name feeling. Several Japanese companies use this system to select appropriate product names (ring and feeling) for new brand products.

3.5.3.4 Type 4: Hybrid Kansei Engineering

In contrast to the forward KE discussed under type 2, type 4 is called Backward KE. This is used in the situation where there is an existing product and the designer wants to know how well this design fits a specific set of Kansei criteria. The computerised system makes design suggestions. If types 2 and 4 are combined then it is called a hybrid KE. (Figure 25).

By means of this type of KE system the designer is able to get design specifications from Kansei words through the forward KE. This helps the designer to be more creative based on his own ideas and suggestions offered by the system. The drawings generated can be input into the system. An image processing system can analyse the sketch. The system is then able to diagnose the input sketch by reference to the Kansei database. This enables the designer to evaluate his own creative design.

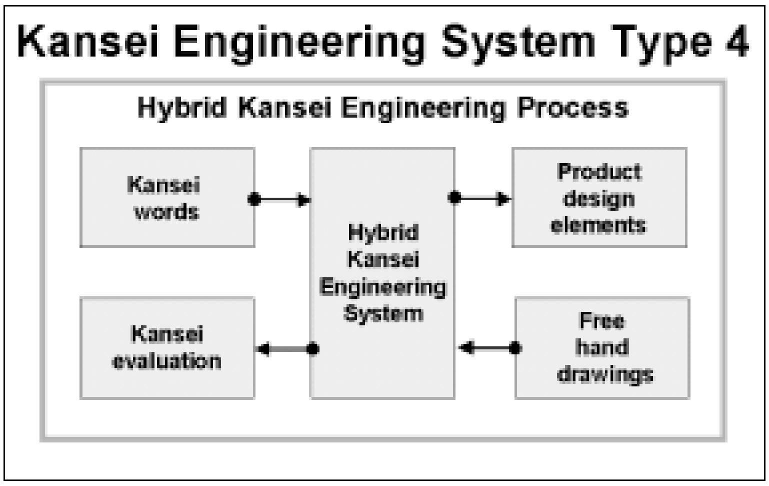

Kansei Engineering System Type 4
Hybrid Kansei Engineering Process
Kansei words
Hybrid Kansei Engineering System
Product design elements
Kansei evaluation
Free hand drawings

Figure 25: Components of a hybrid Kansei Engineering System (Nagamachi 1999)

3.5.3.5 Type 5: Virtual Kansei Engineering

This technique is new and combines KE and Virtual Reality (VR). The advantage of VR is that it enables people to experience computer generated virtual designs by means of a head-mounted display. Control is by means of data gloves. With this technique customers can evaluate the new product that were built by means of KE. Nagamachi constructed a Virtual Kansei Engineering kitchen design system for Matsushita. The kitchen was designed by KE to fit the exact customer needs. Subsequently the customer could evaluate the virtual kitchen by means in terms of his specific Kansei requirements.

3.5.4 Main Kansei Engineering steps

The following main steps are used in a typical KE process. It is a category classification method and is used most often today.

- Clearly define the product purpose.
- Collect the Kansei data using various marketing methodologies.

- Determine the Kansei product mission or baseline. This is the main purpose of the product. In the case of AEDES this could have been "Total Architectural Knowledge Management".
- Break the base line product concept down into primary, secondary and tertiary sub-concepts.
- During the breakdown pay special attention to the appropriate design metrics and issues such as size, mass and material.
- Implement tests such as ergonomic engineering in order to find more detailed design specifications.
- Summarise the overall specifications and review whether they fit the baseline concept.
- Verify the results with the designers' 3D design mock-ups.
- Make adjustments to the final requirements.

3.5.5 The Semantic Differential Method

Advertising and marketing men are frequently faced with the problem of quantifying subjective data with regards the reactions of customers to image of a brand, product or company. In an attempt to solve this problem Snider and Osgood (Snider *et al.* 1957) devised a technique called the Semantic Differential technique (SD). SD attempts to measure what meaning a concept have for people in terms of dimensions which have been empirically defined and factor-analysed. There is a remarkable similarity between this technique and Fuzzy sets (Zadeh *et al.* 1970).

Osgood used a seven-point, equal-interval ordinal scale. These scales were usually selected from 50 pairs of polar adjectives. An example is:

Good						bad

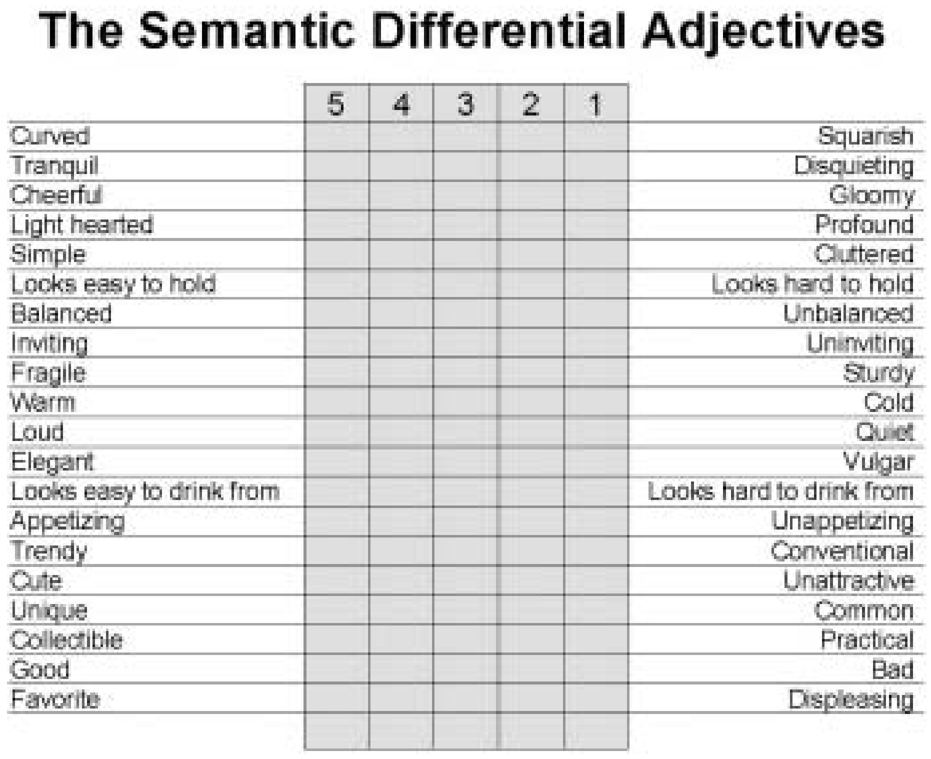
Progressing from left to right on the scale, the positions are described as representing *extremely good*, *very good*, *slightly good*, *being both good and bad*, *slightly bad*, *very bad* and *extremely bad*.

Numeric weights can be assigned to each position. These can be converted to individual or group means and presented in a profile form. The reliability of this method is reasonably high.

The main advantages of SD are:

- It is a quick and efficient way of quantifying large data samples. It captures the direction and intensity of opinions and attitudes towards a concept.
- It provides a comprehensive picture of the image or meaning of a product or personality.
- It is a standardised technique for capturing the multitude of factors which a brand or product comprises.
- It is easily repeatable and reasonably reliable. It can be used on a continuous basis to capture changes in customer attitudes.
- It avoids stereotyped responses and allows individual frames of reference.
- It eliminates some of the problems of question phrasing such as ambiguity and overlapping of statements.

Nagamachi simplified the original Osgood seven-point scale somewhat when he applied it to the design of coffee cups.



The Semantic Differential Adjectives

	5	4	3	2	1	
Curved						Squarish
Tranquil						Disquieting
Cheerful						Gloomy
Light hearted						Profound
Simple						Cluttered
Looks easy to hold						Looks hard to hold
Balanced						Unbalanced
Inviting						Uninviting
Fragile						Sturdy
Warm						Cold
Loud						Quiet
Elegant						Vulgar
Looks easy to drink from						Looks hard to drink from
Appetizing						Unappetizing
Trendy						Conventional
Cute						Unattractive
Unique						Common
Collectible						Practical
Good						Bad
Favorite						Displeasing

Figure 26: Adjectives applicable to coffee cups when using the semantic differential method (Nagamachi 1999)

3.5.5 Conclusion

KE is used in diverse industries such as the automobile, apparel, home appliance, office machinery, home and cosmetics. These industries include some of the most important car manufacturers in the world. Other applications include diverse fields such as digital colour expression, language analysis, video camera and discomfort analysis by means of cross-modality matching. During a workshop attended by the author, Nagamachi stated that although KE is a very comprehensive system it is not a design system, but rather a design support system. KE could certainly be useful to move certain tacit data down to explicit levels where it could be used in design knowledge management and packaging. It is interesting to note the similarities between SD and the dynamic linguistic variables discussed under 3.2.5.7

3.6 Quality Function Deployment (QFD)

3.6.1 Introduction

Due to the important role that a holistic approach to quality plays in the product realisation process this section was included. If architectural design knowledge is to be successfully packaged for use during the product life cycle then quality must form an integral part of it. Figure 2 gives an indication of the tacit level of knowledge that QFD operates at. In the precedent system AEDES (Conradie *et al.* 1999) QFD was made an integral part of the briefing and design system. In the present study QFD was used to analyse the needs of the construction team for a very large construction project. This indicated the need for a design processor clearly. It is now known that QFD is too elaborate for the normal architectural design project. For this reason ARGOS does not form a core part of ARGOS anymore. If it is necessary to use it, it should rather be applied as an outside process. Certain projects might still be suitable for QFD.

The author first learned about Quality Function Deployment (QFD) in May 1998. During further study it was discovered that QFD is one of the most powerful and robust methods to support the product design process. Although QFD was already conceived in Japan in the late 1960s the western world was slow to adopt it. At the moment it is still largely unknown in South Africa and met with scepticism. QFD is an adaptation of some of the Total Quality Management (TQM) tools. The author gained practical experience in the use of QFD during a recent Voice of Customer (VOC) exercise with the members of the professional team to establish accurate user requirements for the AEDES prototype system that is a precedent for the present study.

After World War II statistical quality control (SQC) was introduced to Japan and became the central quality activity in the area of manufacturing. This was integrated with the teachings Juran and Ishikawa. This gradual evolution was strengthened by 1961 publication of Total Quality Control by Feigenbaum. The result of this was that SQC was transformed into TQC during the transitional period between 1960 and 1965. It was at this stage that Akao (1997: 19) became prominent and influential in the subsequent development. Two important factors led to QFD, as we know it today:

- People started to recognise the importance of design quality.
- Companies were already using Quality Control (QC) charts. However the charts were produced during manufacturing after the new products were conceived clearly leaving a quality gap at the initial product conceptualisation.

Akao (1997:19) states that by the time design quality is determined, there should already exist critical Quality Assurance (QA). In 1972 abovementioned deficiencies were addressed in an approach described as "*hinshitsu tenkai*" (Quality Deployment). This established a method to deploy, prior to production start-up, the important quality assurance points needed to ensure the design quality throughout the production process. The method was still inadequate in terms of setting the design quality. This was resolved by means of the quality chart used at the Kobe shipyards of Mitshubishi Heavy Industry. Value Engineering (VE) also influenced QFD. VE is a way to define functions of a product.

In 1978 the term Quality Function Deployment became firmly entrenched. QFD is a literal translation of the Japanese words "*hinshitsu kino tenkai*".

The most important contributions of QFD are (Akao 1997):

- Established quality management in product development and design.

- Provides a communication tool to designers. Engineers and hopefully architects in future are positioned halfway between the market and production. They need to lead product development. QFD gives a powerful means to build a system for product development.
- QFD can significantly contribute to the software industry.
- Future TQM will become important in future to align company-wide activities to customer focus. Akao believes that Voice of Customer (VOC) should be the common bedrock for creating a partnership of such activities.

The Americans are attempting to combine many different ideas with QFD. This includes TRIZ, Taguchi methods and conflict management. QFD and Taguchi methods are gaining attention in the USA as effective methods for concurrent engineering. In product and manufacturing process design, a key optimisation tools is Taguchi's Robust Design Method. The prioritisation capabilities of QFD assist the development team to decide where to apply Taguchi's methods.

The global use ISO 9000 series influenced quality control greatly. It established a global quality standard for the first time. The ISO 9000 series require companies to earn their customer's trust by demonstrating a system of quality assurance. ISO defines a quality system as the organisational structure, responsibility, procedure, process and resource for implementing quality control. Akao predicts that QFD will be recognised as an international standard and be incorporated in ISO.

3.6.2 What is QFD?

QFD is a method for structured product planning and development that enables a development team to specify clearly the customer's wants and needs and then to evaluate each proposed product or service capability systematically in terms of its impact on meeting those needs (Cohen 1995).

The QFD process involves constructing one or more matrices, sometimes referred to as quality tables. The first of these matrices is called the House of Quality (HOQ) (Figure 27). It displays the customer's requirements along the left and the development team's technical response to meeting those needs along the top. The matrix consists of several sections or submatrices joined together in various ways that contains interrelated information.

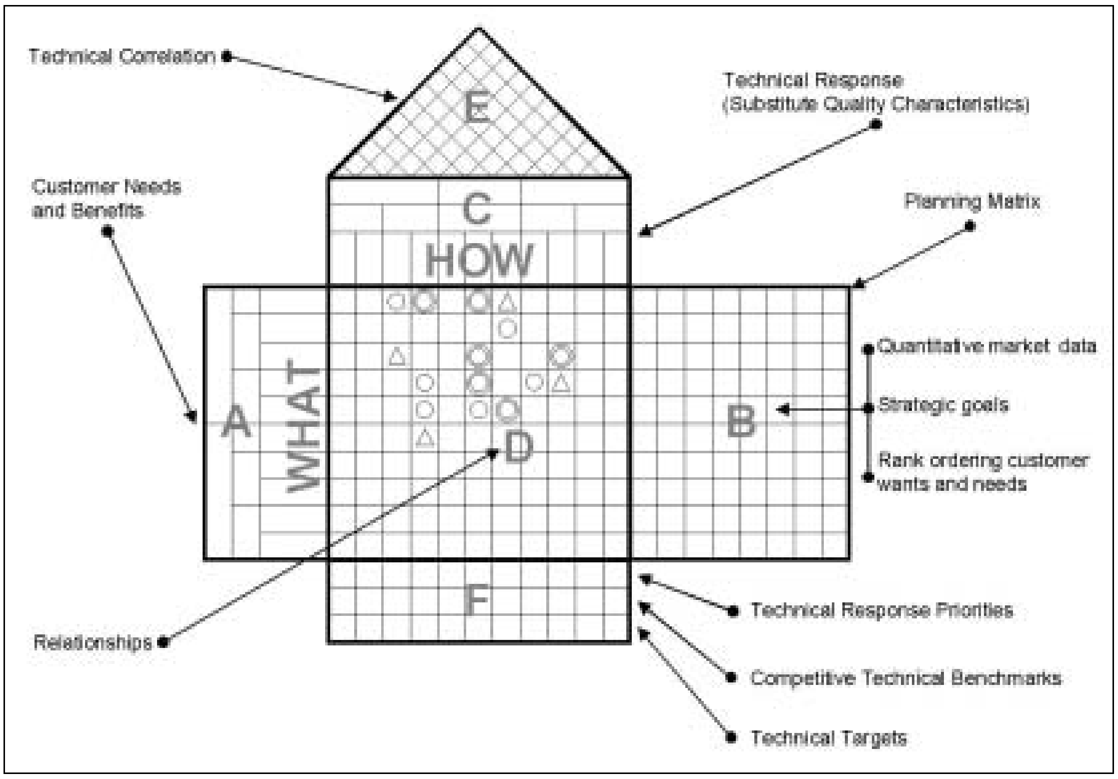


Figure 27: Schematic representation of the QFD House of Quality (Cohen 1995:12)

Each of the labelled sections is a structured expression of the product or process development team's understanding of an aspect of the overall planning process for a new product, service or process.

Section *A* contains a structured list of customer wants and needs. The structure is usually determined by qualitative market research. The data are in the form of a tree diagram that is obtained by methods such as a Voice of Customer exercise.

Section *B* contains three main types of information:

- Quantitative market data. This category consists of three columns that indicate *importance to the customer, customer satisfaction performance* and *competitive satisfaction performance*.
- Strategic goal setting for the new product or service. This category indicates the level of customer performance being aimed for and the improvement ratio required. The two columns normally used here are *goal* and *improvement ratio*.
- A computation for rank ordering the customer wants and needs. Under this main category the ability to sell the product or service, overall importance to the development team of each customer need and cumulative normalised raw weights are normally captured. These three columns are called *sales point, raw weight* and *normalised raw weight.*

Section *C* contains in technical language the description of the product or service they intend to develop. This is normally generated or deployed from the customer wants and needs in section *A*. It is important to note that there will probably not a one-to-one correlation between the user requirements and the technical solutions offered.

Section *D* contains the development team's judgements of the strength of the relationship between the items in *A* and the technical response in *C*. Typically a 9,3,1 scale is used that can also be written by means of special symbols (Figure 27).

Section *E* contains the technical development team's assessment of technical correlation (roof of the quality house) between the items in the technical response.

Section F contains three types of information:

- The computed rank ordering of the technical responses, based on the rank ordering of customer wants and needs from section B and the relationships in section D.
- Comparative information on the competition's technical performance.
- Technical performance targets.

Not all the various sections of the QFD diagram will always be used. In the literature many different variations exist. It is possible to go beyond the initial House of Quality. In the AEDES prototype a system of 5 matrices were used. In this system the HOW of one level of matrix becomes the WHAT at the next level. The effect is that progressive refinement is attained until the desired level of detail is reached.

3.6.3 The affinity diagram

The affinity diagram is a means of for organising qualitative information. The hierarchy is built from the bottom up. The source of ideas into the affinity diagram can be internal or external. The team developing the diagram brainstorms internal ideas. Brainstormed ideas are appropriate for a team that has no data to begin with. In the case of the AEDES Voice of Customer (VOC) exercise an extensive prior literature study was made as to what the requirements might be and also to prepare a structured questionnaire to assist the interviewers in asking the correct questions.

Methods that could be used to hear the VOC are (Technosolve 1998; Cohen 1995):
- Focus group interviews.
- Contextual inquiry.
- Conference room interviews.
- Surveys.
- *Gemba* visits (Observe user in his working environment).
- Walk mile in his shoes.
- Customer complaints.
- Customer requests for existing product enhancement or new products.
- Expert opinion.
- Published sources.
- Social events such as parties or exhibitions.

Table 5: Methods of obtaining Voice of Customer (Collated by author)

	Formal methods				Informal methods				
	Focus group	Contextual inquiry	Conference room interview	Surveys	Gemba visits	Walk mile in his shoes	Customer complaints	Customer requests	Events such as exhibitions
Speed (S,M,F or ~)	F	S	M	~	S	S	F	S	S
Cost (L,M,H or ~)	L	H	M	~	H	H	L	L	M
Requirement yield (L,M,H or ~)	L	H	M	~	M	M	L	~	L

External ideas are the facts that the team acquires. One of the most thorough methods of obtaining data is by means of a customer interview VOC exercise. This takes very careful planning. Questions are prepared beforehand to prepare the interviewers for their task. During an interview attempts are made to uncover as many real unexpressed customer needs as possible. The complete interview is tape-recorded for a subsequent verbatim transcription.

The AEDES development team conducted eight one-hour interviews with the professional team of a large construction project. The example passage below is an example of such a fragment of information that was communicated by one of the interviewees.

In the subsequent analysis the transcribed interviews are very carefully screened to find passages, statements or remarks that clearly express useful thoughts of the customer. The analysis team then hypothesised as to what the statement actually meant and extracted hypothesised user requirements. In the case of the AEDES VOC exercise 173 such user requirements (URs) were identified. On the analysis form careful note was taken of direct product features that are mentioned during the interview that gives direct clues as to desirable product features.

An example of a probing question during a recent VOC interview was:

Q. *" So you can store the process case that you had here and transfer it. From that point of view we already have a bit of a case library that you can draw on in the future?"*

A. *"That is how we go from project to project. I've got a file down there, that has got your process methods and production rates and durations, so that every time you come onto a project…e.g. how many bricks do a bricklayer lay, I've got a set standard that is there. I program the whole programme. I know exactly what should be the duration, then the construction guys go along, they program the whole program and if it is way out, I can say to the guys you are smoking yourselves and it is not from my brain, it is from the files, from the database, it is from the cases that we have put together. That case study is somehow static, however, there is suddenly a new way of laying bricks, then the bricklaying process will change, the duration will change and we will have to update that knowledge that information* *will have to change. But, it is probably quite static at the moment. No new techniques have been developed the last couple of years for brick laying."*

During the subsequent analysis of abovementioned verbatim that clearly expresses a customer need the following hypothesised customer requirements and issues and factors were extracted from the transcription.

Issues/ factors:
- Pre-packaged case histories (experiential knowledge).
- Base information across life cycle process.
- Updating of cases.
- Validation of decisions.

Customer Requirement:
- Enables all team members to refer back to and retrieve experiential project knowledge at any level of grain as and when required.

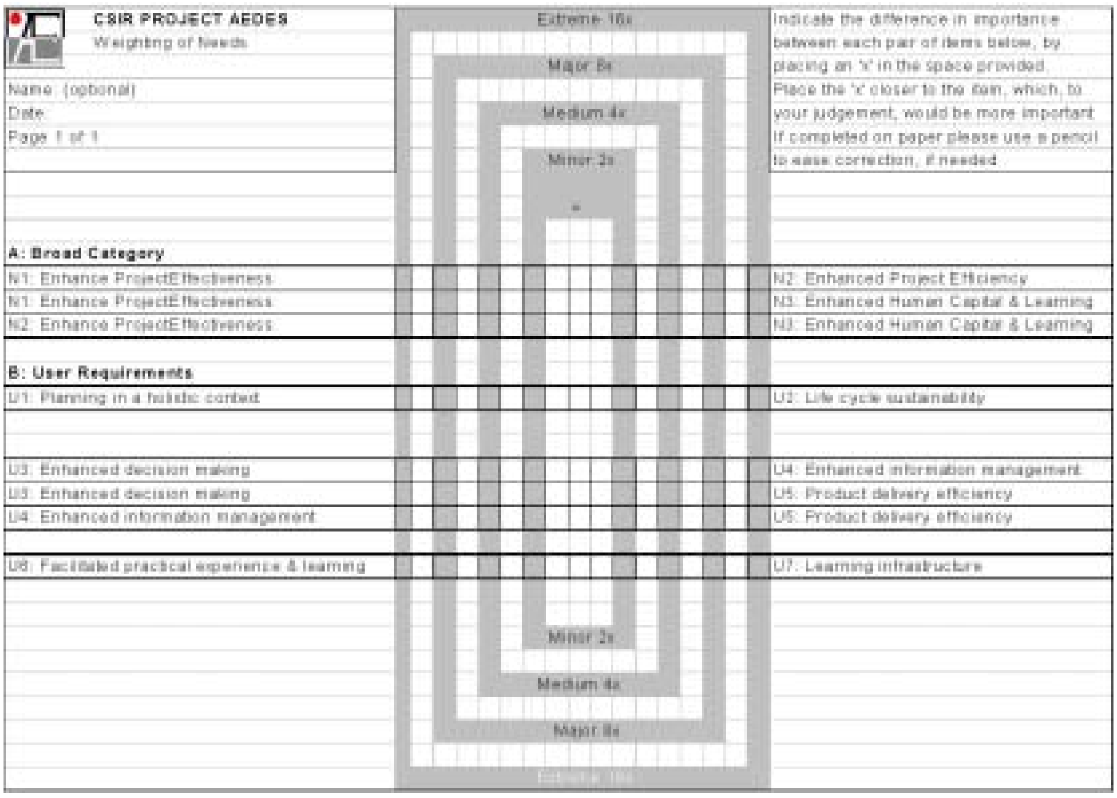
The URs were printed on large sticky labels and pasted onto post-it-notes. The cards were placed on the boardroom table where they could be seen by the entire team. At this stage the team observes total silence and first reads all the cards in sequence in order to get the contents in short term memory. Each member then begins to move the cards together that they think belong together. It is important that the cards are not grouped by similar wording, but rather by similar benefits to the customer. If a card continues to shuttle between different piles the card can be duplicated and placed in two piles. It eliminates a test of wills between two team members.

After the silent sorting process is complete, discussion may start again. The initial 173 URs were grouped into heaps of similar user benefits and a summary title added that best describes the contents of the heap. At this stage the team had 23 abstracted heaps. The next day more effort was put into the grouping exercise and this produced seven higher level URs. At top level it was possible to abstract this to three ultimate user requirements (Table 6).

Table 6: Essential user requirements extracted for the AEDES VOC exercise

Broad Category	User Requirement	Detailed user requirements
Enhance project Effectiveness (Relates to strategy: doing the right thing)	1. Planning in holistic context	1.1 Requirements & methods of life cycle process 1.2 Fast track operational processes 1.3 Contextual visualisation
	2. Life cycle sustainability	2.1 Optimise project in terms of sustainability 2.2 Timeous planning according to type & scale 2.3 Demonstrated financial risk & return
Enhanced project efficiency (Relates to productivity: doing things right)	3. Enhanced decision making	3.1 Sound decision-making 3.2 Co-ordinated decision making 3.3 Trace-ability & validation of decision
	4. Enhanced information management	4.1 Access to information 4.2 Information flows and interchange-ability 4.3 Transparency of interactions & information
	5. Product delivery efficiency	5.1 Problem solving 5.2 Product performance characteristics 5.3 Contractor supply chains 5.4 Quality assurance 5.5 Efficiency in terms of time/cost savings
Enhanced human capital and learning	6. Facilitates practical project experience	7.1 Skills transfer & job creation 7.2 Practical training & learning
	7. Learning infrastructure	7.1 Organisational memory 7.2 Experiential project knowledge 7.3 Rapid access to experiential knowledge 7.3 Generic briefing templates

Table 7 : Sample of form used to extract constant sum paired comparisons from users



CSIR PROJECT AEDES
Weighting of Needs

Name: (optional)
Date
Page 1 of 1

Indicate the difference in importance between each pair of items below, by placing an 'x' in the space provided. Place the 'x' closer to the item, which, to your judgement, would be more important. If completed on paper please use a pencil to ease correction, if needed.

Extreme 16x — Major 8x — Medium 4x — Minor 2x — = — Minor 2x — Medium 4x — Major 8x — Extreme 16x

A: Broad Category

N1: Enhance ProjectEffectiveness	N2: Enhanced Project Efficiency
N1: Enhance ProjectEffectiveness	N3: Enhanced Human Capital & Learning
N2: Enhance ProjectEffectiveness	N3: Enhanced Human Capital & Learning

B: User Requirements

U1: Planning in a holistic context	U2: Life cycle sustainability
U3: Enhanced decision making	U4: Enhanced information management
U3: Enhanced decision making	U5: Product delivery efficiency
U4: Enhanced information management	U5: Product delivery efficiency
U6: Facilitated practical experience & learning	U7: Learning infrastructure

Subsequently a method called "constant sum paired comparisons" was used where each attribute is compared to every other attribute (Cohen 1995:97). This technique is preferred over a five point scoring scales because it yields better statistical accuracy and is non-ordinal. The QFD institute in Detroit also recommends this technique. In this technique a participant is expected to weigh up pairs of factors against each other. This takes careful thinking. The comment is normally that unlike factors are compared, however it must be seen as an importance rating or a rating that tries to determine which factor gives you the most problems. Of the 35 questionaires sent out, a total of nine were returned.

The results of the pairings were analysed and the results indicated clearly the need for a strategic what-if scenario system across the project life-cycle.

The consolidated results produced the results below that are ordered from the most important to the least important.

Table 8: Relative importance of user requirements within group

User requirement	Level of Consensus	Relative importance as a percentage	Normalised relative importance
U2 – Strategic what-if scenarios across project life cycle	Good	30.91	0.31
U1 – Planning in an appropriate holistic context	Good	24.89	0.25
U6 – Practical project experience and learning	Good	16.03	0.16
U7 – Learning support infrastructure during project delivery	Useful	11.47	0.11
U5 – Project delivery efficiency	Useful	7.35	0.07
U4 – Information management across project life cycle	Useful	7.33	0.07
U3 – Decision management across project life cycle	Good	2.02	0.02

The *user requirements* (Customer needs and benefits) can now be filled in on the What side of the QFD matrix. The *relative importance* values can be placed on the planning matrix. The technical team can proceed to generate technical concepts (Technical response or substitute quality characteristics). Analysis of the most important requirements clearly indicates that the following functionalities are required:

- Structured data in appropriate classification containers
- Life cycle software tool modularity
- Systems approach
- Data hierarchies because construction element relationships exhibit a hierarchical structure.
- World wide web connectivity of tools and data
- Data labelling
- Desktop working environment
- Life cycle supply support data like material performance libraries
- Learning support such as intelligent archived case studies that is a complete cognitive snapshot of the various design factors.
- Process support

From this it becomes clear that life cycle process and data integration need to be created. This must further be supported by appropriate modular analysis tools and packaging of designs at various levels of granularity. This is a clear indication that a need for intelligent Case-based Reasoning enabled components exists that could assist the project team with design and operational decisions. This type of component must be flexible to operate in many different software environments at various stages of the product life cycle.

3.6.4 Kano's model of user satisfaction

The Japanese TQM consultant Noriaki Kano provides useful insights of customer satisfaction as it relates to product characteristics. Kano's model divides product characteristics into three distinct categories, each of which affects customers in a different way. The three categories are (Figure 28):

- *Dissatisfiers*. These are "must-be", "basic" or "expected" characteristics.
- *Satisfiers*. These are also known as "one-dimensional" or "straight-line" characteristics.
- *Delighers*. These are also known as "attractive" or "exciting" characteristics.

3.6.4.1 Dissatisfiers

These are product characteristics that the customer takes for granted when they are present, but that cause dissatisfaction when they are missing. Dissatisfiers are things customers do not normally ask for, because they tacidly assume that they will be taken care of. If a product or service is delivered that has many dissatisfiers, customers will be extremely unhappy.

If dissatisfiers are eliminated then customers will hardly notice all the work that has been done to eliminate the dissatisfiers. The reduction of dissatisfiers can only raise customer satisfaction to a "not dissatisfied" state.

3.6.4.2 Satisfiers

This is a feature or characteristic that a customer wants in his product and would usually ask for. The more satisfiers that are provided, the happier customers will be. It is also known as desired quality because it represents the aspects of the product that define it for the customer. Examples of this that were expressed during the AEDES VOC exercise are:

- Compatible data interchange amongst project participants.
- Project specific configuration of software.
- On-line availability of information

In the competitive world of software development one can expect satisfiers to be present in all the competitive products.

3.6.4.3 Delighters

These are product attributes or features that are pleasant surprises to customers when they first encounter them. However if delighters are not present, customers will not be dissatisfied, because they do not know what they are missing. We cannot learn about product delighters by directly asking our customers. Examples of delighters are not as instructive as examples of satisfiers and dissatisfiers. Each delighter is unique and no particular patterns can be identified. Some delighters are entire new products that created entirely new markets. In the present study delighters for the portable design cases called ARGOS could be:

- Integration into any ActiveX compliant container environment such as spreadsheets.
- All pertinent design information available in a convenient to use environment of the users choice.
- Support for design via the Internet.
- Integration of function, shape and quality into a single highly portable mini-case environment.

- The ability to use structured information from the past to assist with future design problems.

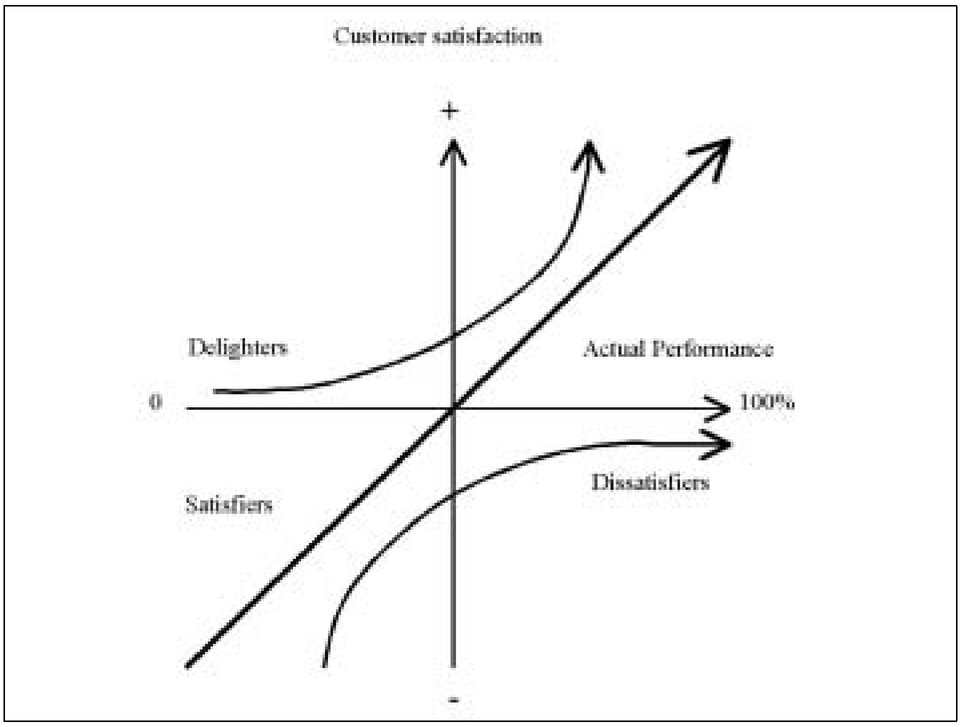


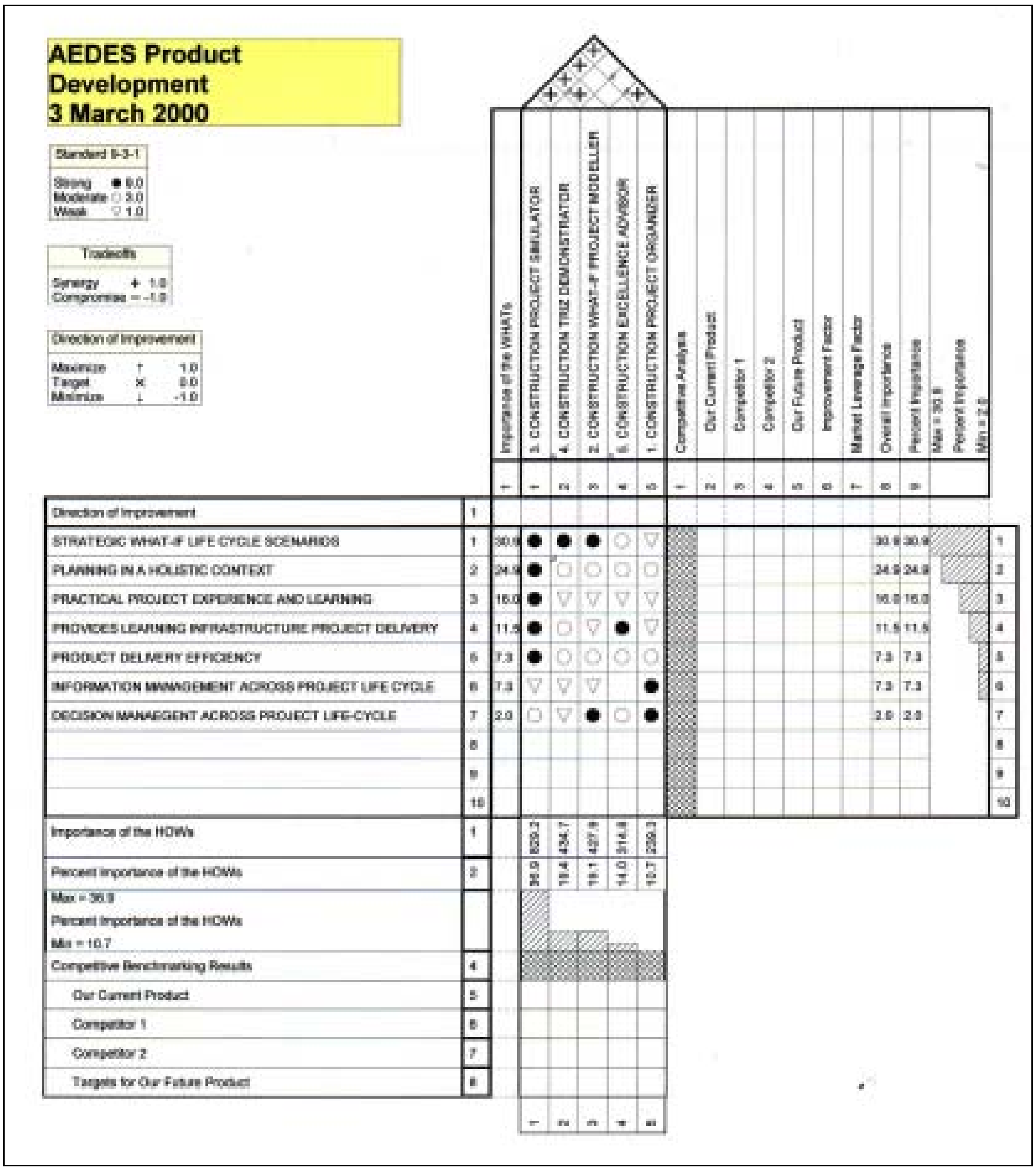
Figure 28: Kano's customer satisfaction diagram (Cohen 1995:37)

The needs that delighters fill are often called latent or hidden needs because they are not directly communicated. QFD offers some assistance in this regard where the interviewers during a VOC exercise attempts to scaffold into the unconscious product desires of the client. During subsequent analysis of the VOC the analysts attempt to cover assumed, expressed and latent elements. QFD is particularly useful because it helps the development team to clearly separate customer needs from technical solutions.

3.6.5 QFD software

Very good commercial QFD software such as QFD/Capture Professional is available. Typical features of this software includes:

- The ability to publish HTML web page output of QFD reports.
- Generate customer surveys in text, Microsoft Word, Rich Text Format and HTML Web Page formats.
- Produce market opportunity map reports identifying the best opportunities for product improvement.
- Generate relationship tree diagrams showing measures for each requirement in a graphical tree and branch format.
- Print out and work with blank chart templates, which are useful as documents-in-progress during team meetings.



AEDES Product Development 3 March 2000

Standard 9-3-1		
Strong	●	9.0
Moderate	○	3.0
Weak	▽	1.0

Tradeoffs		
Synergy	+	1.0
Compromise	−	-1.0

Direction of Improvement		
Maximize	↑	1.0
Target	×	0.0
Minimize	↓	-1.0

		Importance of the WHATs	3. CONSTRUCTION PROJECT SIMULATOR	4. CONSTRUCTION TRIZ DEMONSTRATOR	2. CONSTRUCTION WHAT-IF PROJECT MODELLER	5. CONSTRUCTION EXCELLENCE ADVISOR	1. CONSTRUCTION PROJECT ORGANIZER	Competitive Analysis	Our Current Product	Competitor 1	Competitor 2	Our Future Product	Improvement Factor	Market Leverage Factor	Overall Importance	Percent Importance	Max = 30.9 Percent Importance Min = 2.0	
		1	1	2	3	4	5	1	2	3	4	5	6	7	8	9		
Direction of Improvement	1																	
STRATEGIC WHAT-IF LIFE CYCLE SCENARIOS	1	30.9	●	●	●	○	▽	▨							30.9	30.9	▨	1
PLANNING IN A HOLISTIC CONTEXT	2	24.9	●	○	○	○	○	▨							24.9	24.9	▨	2
PRACTICAL PROJECT EXPERIENCE AND LEARNING	3	16.0	●	▽	▽	▽	▽	▨							16.0	16.0	▨	3
PROVIDES LEARNING INFRASTRUCTURE PROJECT DELIVERY	4	11.5	●	○	▽	●	▽	▨							11.5	11.5	▨	4
PRODUCT DELIVERY EFFICIENCY	5	7.3	●	○	○	○	○	▨							7.3	7.3	▨	5
INFORMATION MANAGEMENT ACROSS PROJECT LIFE CYCLE	6	7.3	▽	▽	▽		●	▨							7.3	7.3	▨	6
DECISION MANAEGENT ACROSS PROJECT LIFE-CYCLE	7	2.0	○	▽	●	○	●	▨							2.0	2.0		7
	8							▨										8
	9							▨										9
	10							▨										10
Importance of the HOWs	1		829.2	434.7	427.9	314.8	239.3											
Percent Importance of the HOWs	2		36.9	19.4	19.1	14.0	10.7											
Max = 36.9 Percent Importance of the HOWs Min = 10.7			▨	▨	▨	▨												
Competitive Benchmarking Results	4		▨	▨	▨	▨	▨											
Our Current Product	5																	
Competitor 1	6																	
Competitor 2	7																	
Targets for Our Future Product	8																	
			1	2	3	4	5											

Figure 29: QFD/Capture product planning matrix screen (Author)

During the prototype development of the AEDES software the author developed QFD software that could integrate the architectural briefing and design process directly in an underlying database. This was an attempt to make the information captured during the QFD briefing and design sessions directly available to other distant members of the design team. The biggest difference between the AEDES QFD and standard software is the fact that it worked in depth as well. In-depth implied that the user could see more detail by clicking on the intersection of a specific row or cell (Figure 31).

QFD has a practical limitation in the sense that it cannot conveniently accommodate more than a 20 by 20 matrix. Architecture contains information at many different levels of detail that is likely to give rise to very large matrices. This was solved by means of the in-depth method. The disadvantage of the latter is that it is not possible to see information directly at a glance. However reports were developed that can be printed out and studied at leisure.

The in-depth method required a special database structure that is detailed in Figure 31. The relational database table *QFD* contained the main QFD project information. Two main hierarchies branch from this main table, i.e. the *QFDHow* and *QFDWhat* branches each having respective subtables called *QFDSubHow* and *QFDSubWhat*. Special connecting tables (somewhat unusual in relational database design) were used to keep book of the relationships between the What and How data branches. The relationships are what would occur in section D (Figure 27). The tables *QFDRoof* and *QFDSubRoof* are self-referring tables (recursive) and are designed to support the relationships that are required by the QFD technical correlations (QFD roof).

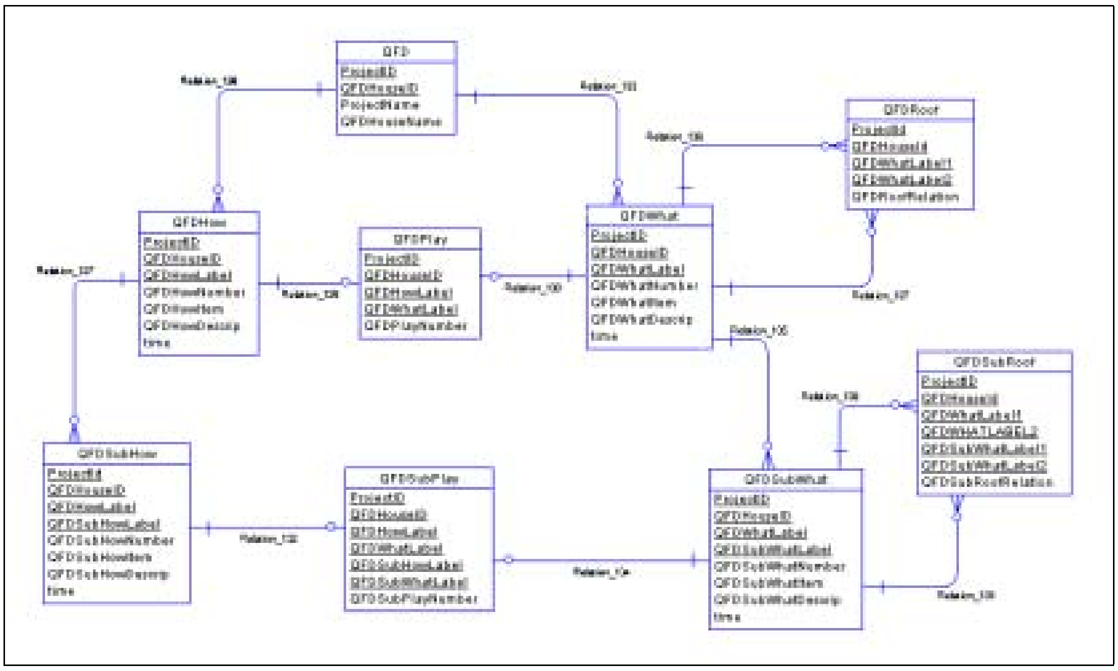
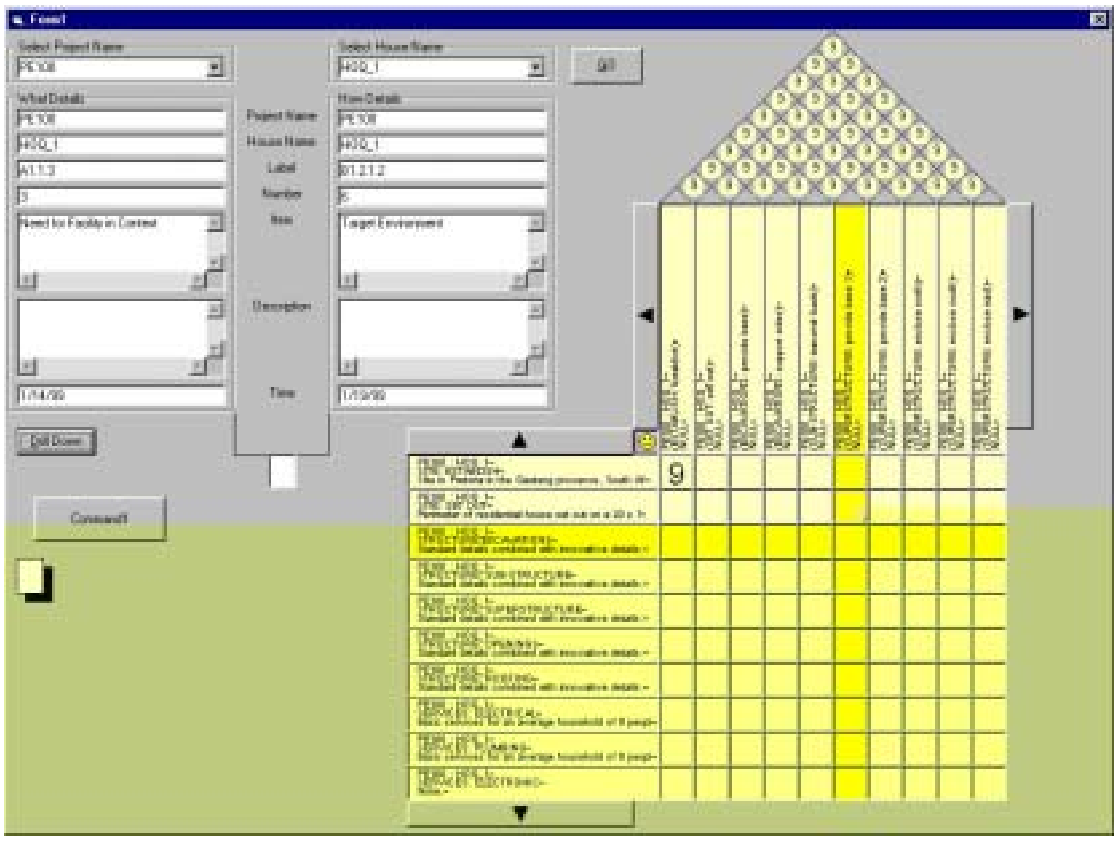


Figure 30: Relational database tables used in the AEDES prototype QFD software (Author)

The software only allowed viewing of the design data. Editing was accomplished by means of special database forms. The QFD software provided a convenient means to view the numerous different technical correlations that exists in architecture. The software provided convenient navigational command buttons that facilitated navigation across a larger than displayed virtual QFD matrix. A drawback of this was that it was not possible to view the entire matrix at a glance. The author is of the opinion that it is not always necessary to see all design issues at once in architecture, because not all design factors at all levels are so closely related that it is necessary to have simultaneous visual display. Future improvement of the software could be to write QFD software in a Visual Basic ActiveX control. This would greatly improve the usefulness of the QFD software because it would then be possible to use the advanced methodology in a convenient environment such as a spreadsheet, CAD systems or it could be integrated into software shells developed in languages such as Visual Basic or Visual C++.



What Details		How Details
PE108	Project Name	PE108
HOQ_1	House Name	HOQ_1
A1.1.3	Label	B1.2.1.2
3	Number	6
Need for Facility in Context	Item	Target Environment
	Description	
1/14/99	Time	1/13/99

Figure 31: Typical screen of the AEDES prototype QFD software (Author)

3.7 Theory of inventive problem solving (TRIZ)

3.7.1 Introduction

There are two groups of problems people face, those with generally known solutions and those with unknown solutions. Those with known solutions can usually be solved by information found in the technical literature or through extensive training. These solutions follow the general pattern of problem solving. Here a standard solution is elevated to a standard problem of a similar or analogous nature. A standard solution is known and from that standard solution comes a particular solution to the problem.

The other type of problem has an unknown solution. It is called an inventive problem and may contain contradictory requirements. In modern times inventive problem solving falls in the field of psychology where the links between the brain, insight and innovation are studied. Methods such as brainstorming and trial-and-error are commonly suggested. Depending on the complexity of the problem, the number of trials will vary. If the solution is within the field of experience then the number of trials will be fewer. If the solution is not found the inventor must look beyond his experience and knowledge to new fields such as manufacturing or aviation. Then the number of trials will grow large depending on how well the inventor can master psychological tools like brainstorming, intuition and creativity. A further problem is that psychological tools like experience and intuition are difficult to transfer to other people in the organisation.

This leads to what is called psychological inertia where the solutions being considered are within the inventor’s own experience and do not consider alternative technologies to develop new concepts. When we overlay the limiting effects of psychological inertia on a solution map covering broad scientific and technological disciplines the ideal solution might lie outside the inventor’s field of expertise. Psychological inertia defeats randomness and leads to looking only where there is personal experience.

3.7.2 TRIZ

Genrich S. Altshuller, born in the former Soviet Union in 1926, developed a superior approach relying on technology. His curiosity about problem solving led him to search for standard methods. Altshuller screened over 200 000 patents looking for inventive problems and how they were solved. Only 40 000 had somewhat inventive solutions, the rest were straightforward improvements. At this stage it is estimated that more than a 1 000 000 patents have been screened world-wide. Altshuller defined an inventive problem as one in which the solution causes another problem to appear. Usually inventors must resort to a trade-off and compromise between the features and thus do not achieve an ideal solution. In his study of patents he found that many described a solution that eliminated or resolved the contradiction and required no trade-off. Altshuller identified five levels of inventive solutions (Kaplan 1996:2; Mazur 2001):

- Level one. These are routine design problems solved by methods well known within the speciality. No invention is required. About 32% of the solutions fell into this level.
- Level two. These are solutions that leave the existing system fundamentally unchanged. New features are introduced or minor improvements are made to the existing system. This is effected by known methods and sometimes compromises may be made. About 45% of the solutions fell into this level.
- Level three. This constitutes an essential improvement of an exiting system. Methods outside the known industry are used. Certain contradictions need to be resolved. About 18% of the solutions fell into this category.

- Level four. At this level inventions are characterised by solutions found in more in science than in technology. Only about 4% of the solutions fell into this category.
- Level five. This is the level where rare scientific discoveries or pioneering inventions occur. Only about 1% of the solutions fell into this category.

He also noted that with each succeeding level, the source of the solution required broader knowledge and more solutions to be considered before an ideal one could be found. Altshuller found that 90% of the problems engineers faced had been solved somewhere before. If engineers could follow a predictable through the various levels and using their knowledge and experience most of the solutions could be derived from knowledge already present in the particular company or industry.

Altshuller distilled the problems, contradictions and solutions to these patents into a comprehensive theory of inventive problem solving which he named TRIZ.

There are a number of laws in the theory of TRIZ. One of them is the law of Increasing Ideality. A technical system evolves in such direction as to increase its degree of Ideality (Kaplan 1996). Ideality is defined as the quotient of the sum of the system's useful effects, U_i , divided by the sum of its harmful effects, H_j .

$$Ideality = \frac{\sum U_i}{\sum H_j}$$

Useful effects include all the valuable results of the system's functioning. Harmful effects include undesired inputs such as cost, the space occupied, energy consumed, pollution and danger. The ideal state is one where there are only benefits and no harmful effects also termed the *Ideal Final Result*. From a design point of view, engineers must continue to pursue greater benefits and reduce cost of labour, materials, energy and harmful side effects. If the improvement of a benefit results in increased harmful effects, a trade-off is made, but the Law of Ideality drives designs to eliminate or solve any trade-offs or design contradictions. The ideal final result will eventually be a product where the beneficial function exists but the machine itself does not. The evolution of the mechanical spring-driven watch into the electronic quartz crystal watch is an example of this move towards Ideality.

Boris Zlotin and Alla Zusman, TRIZ scientists at the American company Ideation and students of Altshuller have developed an "Innovative Situation Questionnaire" to identify the engineering system being studied, its operating environment, resource requirements, primary useful functions, harmful effects and ideal result.

3.7.3 Steps in using TRIZ

3.7.3.1 Formulate the problem: the prism of TRIZ

This first step is to restate the problem in terms of physical contradictions. Identify problems that could occur. Could improving one technical characteristic in solving the problem cause other technical characteristics to worsen, resulting in secondary problems? Are there technical conflicts that might force a trade-off?

3.7.3.2 Search for previously well-solved problems

Altshuller extracted from over 1 500 000 worldwide patents 39 standard technical characteristics that cause conflict. These are called the 39 Engineering Parameters. Find the contradicting engineering principles. First find the principle that needs to be changed. Then find the principle that is an undesirable secondary effect. State the standard technical conflict.

3.7.3.3 Look for analogous solutions and adapt to solution

Altshuller also extracted from the worldwide patents 40 inventive principles. These are hints that will help an engineer find a highly inventive (patentable) solution to the problem. To find which inventive principles to use, Altshuller created the table of Contradictions. This table lists the 39 Engineering Parameters on the X-axis (undesired result or conflict) and Y-axis (feature to change or improve). The appropriate Inventive principles that could lead to a solution are listed in the intersecting cells.

3.7.3.4 Socially responsible TRIZ

Structurally and philosophically TRIZ methods look at the big picture. During problem definition the TRIZ practitioner looks at nine combinations of the past, present and future models of the sub-system, system and super-system. Interactions, resources, harmful and secondary effects are identified during the definition of the problem.

Terninko (1999:285) states that the TRIZ method can support sound environmental design through the recognition of resources within the sub-systems, systems and super-systems. If the future of the system and super-system is well understood then possible future disastrous effects can be avoided. He suggests a different type of TRIZ formula to take account of the harmful effects.

$$Ideality = \frac{\sum benefits}{\sum \cos ts + \sum harms}$$

The equation above is more a construct than a directly usable equation. This particular version of the ideality equation contains cost and harmful effects in the denominator.

It is not difficult to identify the benefits and this is what the TRIZ specialist normally tries to understand. The identification of possible harmful effects and its alternatives is just as important. The product designer is normally far too casual about the costs and harms in the denominator. See item 4.2.1.2 for Sustainable Development.

TRIZ does represent a method that can be socially responsible, but the practitioner must resist pressure from society and industry for rapid and incomplete analysis. Organisations are often driven by profit while ignoring the customer and the medium to long consequences of their solutions.

Summary

The techniques and product innovation methodologies discussed in this chapter are useful at various tacit and explicit levels. They are also applicable in different building life cycle phases (Figure 2).

It is clear that after the initial optimism about the possibilities of AI in design, a more mature and realistic approach is now followed. CBR is a promising sub-field of AI that can greatly contribute to the contextual storing of design knowledge. It is clear that AI should be used more in the background and especially in architecture automatic adaptation of designs should not be attempted.

Knowledge Management is becoming very prominent although there are still unsolved problems. However many researchers are working on the particular sub-problems due to the importance of this for the global economy. KM is still fluid, however the theory is well understood such as the movements of the knowledge cycle. The sharing of knowledge is important in any enterprise and this is supported by the current importance attached to intellectual capital. *Concept extraction* and *Natural Language Processing* remains problematic, however significant progress has already been made. The current and emerging technical standards that form a barrier to responsive NGM were identified. The main requirements for a KM enabling environment are:

- Communication
- Design team flexibility and responsiveness
- User Interface and information search
- Project resource integration and access

The problems of ontology and the role that AI can play in Knowledge Based Design were investigated. It was observed that CBR and the concept selection cycle of Pugh (1996) bear striking similarities. The various main known problem-solving architectures were investigated and the conclusion can be made that CBR, RBR and MBR should be not be seen in isolation but should rather be viewed as a continuum of techniques.

The use of fuzzy sets as a means of formulating dynamic linguistic variables for aiding the retrieval of design knowledge in general and cases specifically were investigated. It was discovered that the semantic differential method of Snider and Osgood (Snider *et al*. 1957) and the semantic differential adjectives as used by Nagamachi bear a relationship to the approach advocated by the author.

The analysis of the characteristics of manufacturing such as process, flow and throughput indicate that these are not directly applicable to problems under consideration, but should rather be applied at the process level. Concurrent Engineering is an important technique to avoid the so-called time-trap. This is where the life cycle time of products decreased while the time spent on product development greatly increased. Three possible strategies could be identified as CE guiding principles:

- Parallelisation.
- Standardisation
- Integration

The theories of Goldratt showed that the manufacturing environment is particularly applicable for Theory of Constraints and that the system optimum is not the sum of the local optima.

Taguchi techniques indicate the importance of off-line and on-line quality control. This indicates that quality is related to the loss to society caused by a product during its life cycle. In terms of the current thesis these methods should rather be used select appropriate materials to minimise life cycle costs in the context of sustainable development.

The advantages of the fuzzy Front End (FFE) was identified:

- It lasts a long time
- Cheap place to look for cycle time
- Individual companies have big performance differences

The investigation of objects indicates that it is the preferred way to achieve abstraction, generalisation and interaction in systems supporting the life cycle development process. The unique way those objects were used in the precedent systems PREMIS and AEDES were discussed. This section was concluded with strategies followed by Microsoft to establish if these could offer opportunities for the packaging of architectural design and design parameters.

Kansei Engineering (KE) is a mature and useful technique to quantify cognition and product image in such a way as to influence the product development process. KE operates at a very high tacit level is could make a significant contribution to the storage of tacit architectural design information.

The QFD exercise undertaken indicated that the ability to generate what-if scenarios across the project life cycle as the most important user requirement. QFD as a technique to extract raw architectural user requirements was pioneered in the AEDES system. QFD is useful if the time and cost can be justified. QFD is an important technique in the manufacturing industry and was one of the techniques that saved the Detroit automotive industry from ruin in the face of severe competition from Japan. The contribution that QFD can make in architecture is dependent on the acceptance that this slightly elaborate technique can gain.

The chapter is concluded with TRIZ that is a powerful method to solve inventive problems. However the present commercial TRIZ software emphasise engineering type of problems. A significant amount of work will have to be done to make its use tractable in architecture.