Studies in Systems, Decision and Control 51

Staffan Sunnersjö

Intelligent Computer Systems in Engineering Design

Principles and Applications

Springer

Studies in Systems, Decision and Control

Volume 51

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
email: kacprzyk@ibspan.waw.pl

About this Series

The series "Studies in Systems, Decision and Control" (SSDC) covers both new developments and advances, as well as the state of the art, in the various areas of broadly perceived systems, decision making and control- quickly, up to date and with a high quality. The intent is to cover the theory, applications, and perspectives on the state of the art and future developments relevant to systems, decision making, control, complex processes and related areas, as embedded in the fields of engineering, computer science, physics, economics, social and life sciences, as well as the paradigms and methodologies behind them. The series contains monographs, textbooks, lecture notes and edited volumes in systems, decision making and control spanning the areas of Cyber-Physical Systems, Autonomous Systems, Sensor Networks, Control Systems, Energy Systems, Automotive Systems, Biological Systems, Vehicular Networking and Connected Vehicles, Aerospace Systems, Automation, Manufacturing, Smart Grids, Nonlinear Systems, Power Systems, Robotics, Social Systems, Economic Systems and other. Of particular value to both the contributors and the readership are the short publication timeframe and the worldwide distribution and exposure which enable both a wide and rapid dissemination of research output.

More information about this series at http://www.springer.com/series/13304

Staffan Sunnersjö

Intelligent Computer Systems in Engineering Design

Principles and Applications

Springer

Staffan Sunnersjö
Jönköping University
Jönköping
Sweden

ISSN 2198-4182 ISSN 2198-4190 (electronic)
Studies in Systems, Decision and Control
ISBN 978-3-319-28123-0 ISBN 978-3-319-28125-4 (eBook)
DOI 10.1007/978-3-319-28125-4

Library of Congress Control Number: 2015958837

Printed on acid-free paper

This Springer imprint is published by SpringerNature
The registered company is Springer International Publishing AG Switzerland

To my wife, Gertie Sunnersjö

Preface

We have all seen street musicians at work playing several instruments simultaneously and admired their ability to sound like an orchestra single handed. These somewhat unusual musicians struggle with many and diverse instruments trying to produce well-orchestrated music that the audience will find pleasing enough to contribute to the artist's bread-and-butter. To succeed, the performing artist must be proficient with many instruments, be able to improvise as he goes along and have an ear for the total effect of his efforts. The practitioner of engineering design might feel a certain kinship with this man. Like the musician, the engineering designer needs to have skills in many technical disciplines, to be able to improvise and to come up with new ideas as problems are encountered. Further, in the same way as the music, the product that comes out of the design process will be judged in its entirety, not by the properties of its individual parts.

Is it meaningful, or even possible, to use computers for unstructured, multidisciplinary and creative tasks such as engineering design? This book seeks to give a qualified answer to this question. It will be apparent that the simple question has a complex answer.

There are many tasks in engineering where computers have established a significant role often concerned with product documentation or analysis. This book however is directed towards the use of computers for the purpose of creation. It focuses on how to simulate the work of a design group by implementing engineering knowledge in a computer and then letting the computer operate on this knowledge to produce new design solutions. This is not the kind of creative tasks we normally expect computers to be able to perform and it might appear as if the computer possessed some kind of intelligent behaviour, a kind of Artificial Intelligence. The resemblance to human intelligence is however superficial, but the technology has proved its worth on many types of design tasks. We call this technology Automated Engineering Design or Design Automation, DA. The focus is on the synthesis tasks of design work rather than the analysis tasks, which are normally aimed at predicting the behaviour and properties of a given design proposal.

In this book, the term Automated Engineering Design is used in the straightforward meaning of partly or completely automating tasks in the engineering design process by the use of computer support. Such tasks are mainly found within a category of what is called “variant design” or “redesign” in a broad sense. This implies that the intended design process is well understood and does not require any innovations or new knowledge—the computerised design system will always operate “inside the box”.

Designing new products is a different matter. This will always require elements of creativity, resourcefulness and subjective judgment. Good, original design must exhibit a “spark of life” to be successful on the market. This, in the opinion of the author, is something a computer will never be able to deliver. Admittedly though, past prophecies of the future of the digital computer have in retrospect tended to underestimate its potential.

This book aims to discuss how to plan and build useful, reliable, maintainable and cost-efficient computer systems for automated engineering design. The book takes a user's perspective and seeks to bridge the gap between texts on principles of computer science and the user manuals for commercial design automation software. The approach taken is top-down, following the path from definition of the design task and clarification of the relevant design knowledge to the development of an operational system well adapted for its purpose.

This is an introductory text for the practicing engineer working in industry and covers most vital aspects of planning such a system. Experiences from applications of automated design systems in practice are reviewed based on a large number of real, industrial cases. The principles behind the most popular methods in design automation are presented with sufficient rigour to give the user confidence in applying them on real industrial problems. The textbook is also suited for a half semester course at graduate level and has been complemented by suggestions for student assignments. The primary purpose of the book is to discuss application of design automation methods, but the reader will find that the study of design automation will also shed new light on the manual design process as is often the case when computer technology is applied to a familiar task.

The perspective is that of the specialist of the field of application rather than the specialist in computer science. For the latter, who has an interest in seeing his/her subject from a user perspective, the book will offer new insights. The examples are taken from the field of mechanical engineering, but the book should be equally applicable to, e.g. civil engineering or electrical engineering. The book has grown out of the lecture notes of two postgraduate courses given annually or biannually during the last ten years at the product development program at the School of Engineering at Jönköping University.

Some time ago, a young relative of mine asked me what my research work was all about. I said that I worked with computer support for variant design and explained in a few words what this meant. She replied: “But Staffan, this sounds incredibly boring, how can you stand working with such things?” I was somewhat taken aback by this reaction since I have always found my work exciting and

thought-provoking. Obviously I had not succeeded in communicating my enthusiasm for the subject. When writing this book, I have tried to keep this conversation in mind in order to not only cover the utilitarian aspects of the subject but also emphasise its much wider implications and the insights it gives into designing and planning and, maybe, into human thinking in general.

Jönköping, Sweden Staffan Sunnersjö

Acknowledgements

This book tries to summarise more than 20 years of experience of research, teaching and application of Design Automation methods carried out at the Swedish Institute of Production Research, IVF, and later at the department of Mechanical Engineering at the School of Engineering, JTH, Jönköping University, Sweden. The author is indebted to colleagues and management at these two research establishments for providing funds and facilities and a stimulating environment. I would like to express my gratitude to Ingvar Rask and Rafael Amen at IVF for the expertise they have contributed to our research in the field. At JTH, discussions both on the development of Design Automation know-how and how to teach the subject have been many and inspiring. Special thanks are due to Fredrik Elgh, Mikael Cederfeldt, Joel Johansson and Roland Stolt, all at JTH. Several of the cases of application in this book originate from the works of these colleagues. I am also indebted to a group of specialists active in industry, notably Esa Ryhanan at E-Rules and Sandvik Coromant, who have shared generously of his own experiences and made valuable comments.

Finally I would like to express my gratitude to my lovely wife Gertie, who has supported me in this work and also proof-read the manuscript. I do not think she has any particular interest in the subject, but as always, she has encouraged me and made my life pleasant during the literary labour, reminding me that there are other things in life than engineering. For this I dedicate this book to her.

Jönköping, Sweden Staffan Sunnersjö

Contents

About the Author

Staffan Sunnersjö is a professor emeritus in Machine Design at the School of Engineering at Jönköping University, where he has lead a research group in computer-supported engineering design. He was previously head of the division for Engineering Design at the Swedish Institute for Production Research and before that worked as a consultant and in industry at various R&D positions mainly related to shipbuilding and electrical power generation for about 15 years.

After receiving his Ph.D. in Mechanical Engineering at the University of Aston in Birmingham, UK, in 1976, he specialised in structural dynamics and vibration control until the early nineties when his research focus turned towards engineering design methodology and design automation. He has published four books and more than fifty scientific papers about engineering design and taught postgraduate courses on the subject, but also has considerable practical experience of development and implementation of design automation systems in industry.

Chapter 1
Introduction

1.1 Design of Industrial Products

As humans we have used man-made objects since the dawn of times, but it is only since the industrial revolution some 250 years ago that the cost efficient engineering materials and manufacturing processes emerged that have since fundamentally changed our society. Industrial production is often characterised by mechanisation, large scale operations, utilisation of mechanical power, specialisation and standardisation.

Engineering design is about planning products that are to be manufactured by an industrial process. In order to maintain their efficiency, these manufacturing processes impose certain constraints on the product design. The constraints could be of a purely technical nature depending on the intended manufacturing process, or be due to company policies or be economically motivated. These are requirements imposed by the manufacturing company, but the product must also be found attractive on the marketplace. A set of requirements that will make the products interesting and desirable for prospective customers must thus also be met. Balancing the internal company requirements against the expectations and needs of the market is a core issue for all designers and the competitiveness of a new product relies very much of how skilfully and resourcefully these, often conflicting, requirements are met.

One aspect of this is the diversity of product variants. The potential buyer would be most pleased if all products were tailor-made to suit the needs of the individual customer. The manufacturing and logistics departments of the supplier on the other hand, would prefer very long series of standard products and the production of any "specials" is usually strongly opposed. This is a consequence of the industrial process: Crafts products are made individually and by hand, industrial products are manufactured in series in a standardised and efficient process.

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_1

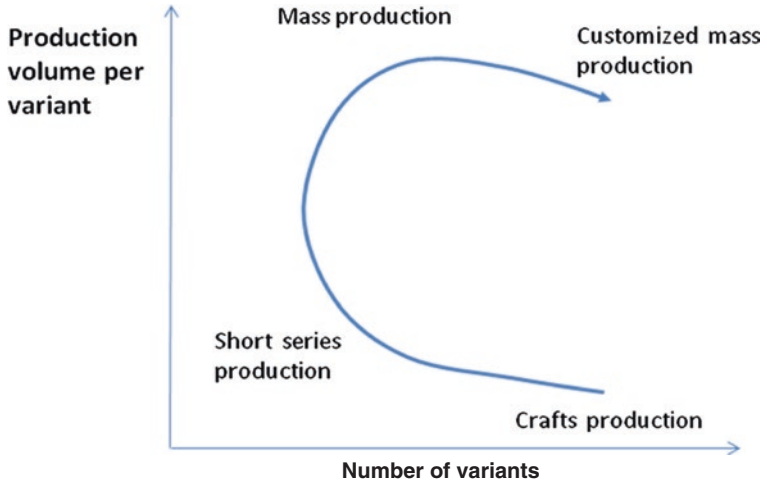


Fig. 1.1 Long term change of production paradigm. From an idea by Womack (1990)

Standardisation is one aspect of the term "industrial production" which needs to be somewhat elaborated before we proceed.

The strive for *standardisation* is typical for industrial production and is also a necessity for the automation of design processes. This applies to the standardisation of the product itself, i.e. its parts, modules and product structure, as well as the standardisation of the design procedures used. Standardisation inevitably implies restrictions of design freedom. This might appear detrimental to the usefulness of the method, but very often practical experience shows that this side effect can be highly beneficial. Properly applied measures to bring about standardisation reduce costs for inventory, logistics and manufacture. Industrial efficiency comes with repetition.

It is thus natural that crafts products are easily tailored to the requirements of individual customers, while products coming out of an industrial process are more standardised. When using automated engineering design systems there is an overall ambition to *combine the flexibility of piecewise production with the efficiency of mass production*. Figure 1.1 illustrates this shift in long term production paradigms from crafts products, over series and mass production of standard articles towards mass production with a high degree of variety for different markets and customers—*mass customisation*.

Although application of design automation methods is a necessity when supplying customised products in an industrial scale, it is by no means the only requirement. There is also a need for a well-planned flexibility of component designs, product structures and production processes. Seen in this perspective design automation belongs to the core technologies in design and production of industrial products.

1.2 Which Are the Motives for Design Automation?

Engineering design is creative work. It requires ingenuity, intuition and good judgement—all talents that are typically human and not easily implemented in a computer program. This might seem discouraging from an automation perspective and so it is as long as we are talking about original design. Investigations of how time is spent at industrial design offices, see e.g. (Encanação et al. 1990; Ullman 1997) or Cederfeldt and Elgh (2005), indicate however that typically 70–90 % of working hours are spent modifying, adapting or in some other way redesigning already existing and proven, basic design solutions. In Crabtree (1993) it is reported from interviews with designers in the US aircraft industry that 56 % of product development delays were due to lack of documented design knowledge. This underlines the importance of reuse of ideas and engineering knowledge that is established and form part of a company's *corporate knowledge*. Redesign work is by no means trivial, but it does not rely on creativity, but rather on skilful application of existing product knowledge. This is the natural field of application for design automation methods and this is where the major benefits usually are found.

Product development traditionally aims at a basic product design that is viable on the market. The key to real economic success however often lies in the creation of a *product family* based on this original design. The technology of the basic design is reused for the whole range of product variants with a minimum of cost, thereby multiplying the benefits of the original investment. This means that creation of product variants for a better penetration of established markets and gaining access to new markets becomes possible without major investments in development projects.

Design automation offers opportunities to increase speed, quality, cost effectiveness and other desirable characteristics of product variant design. There is also a trend among customers to discard standard products and demand a higher degree of customisation. The process of globalisation also requires products to be made in many versions for different countries and markets. Computers are very efficient for tasks involving laborious, repetitive work, while humans excel in tasks requiring creativity, judgment, common sense and other similar traits. It is of critical importance to distinguish which engineering tasks that might successfully be carried out by a computer and which tasks are better handled by a human designer. In fact one strong motive to introduce design automation systems is to release engineering resources for creative work, e.g. original design. Although the use of a DA system often requires no creativity, the setting up of one is a highly creative and demanding task.

Successful companies have often created an element of uniqueness in their business processes. This applies also to their use of design automation methods as will be apparent in the industrial examples in Appendix A. Nevertheless, the

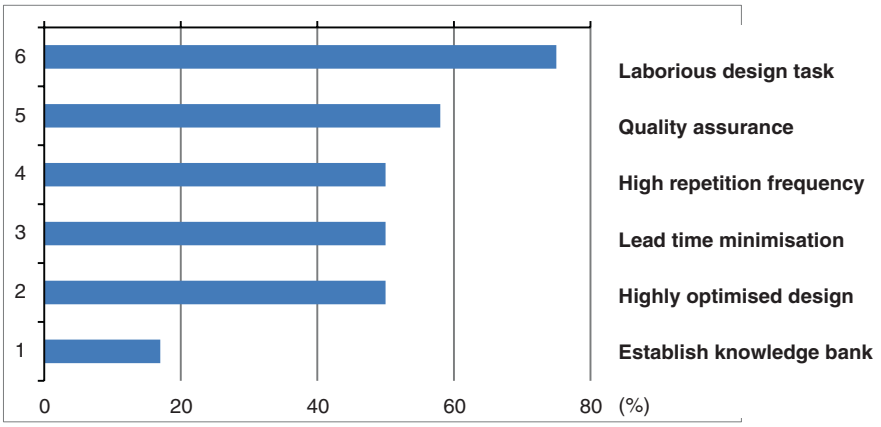


Fig. 1.2 Distribution between six common motives for automating variant design at twelve manufacturing companies. Drawn from results in Amen et al. (1999)

following motives are often found when analysing what benefits companies gain by using design automation methods:

- Dramatic reduction of lead times in design, analysis, process planning, cost estimation and production preparation,
- Relieve designers and production planners of routine work and make resources available for creative tasks,
- Better optimisation of products and processes,
- Quality assurance—Streamlined, stringent and traceable development process that is not critically dependent of which individual that is doing the design work,
- Acquisition and accumulation of corporate knowledge.

In a survey of twelve Swedish manufacturing companies either using or in the process of planning a Design Automation system, the objectives for applying this method was investigated. The distribution between six common objectives is shown in Fig. 1.2. The least common objective is the establishment of a corporate knowledge bank. In the opinion of the author, companies underestimate this aspect, which might well be the most important factor. The value of the knowledge bank might however be significant in the long term perspective, while companies often have to achieve very quick return on invested time and money.

1.3 Modelling and Simulation of the Design Process

Design automation deals with the tasks of idealising, modelling and simulating the design process and its associated knowledge. When a new product variant is required the computer model is executed with the new specifications as input

resulting in an adapted product variant as output. The computer model can be seen as a *simulator* that emulates the designer or the design team.

Modelling is a key concept in many branches of engineering science. The purpose of using models is to create a simplified description of a phenomena or a process that if completely represented becomes too detailed and unwieldy to be practical. In the classical engineering disciplines models are usually of a mathematical nature and are adjusted and verified by comparisons with experiments. Such models are used to *analyse* product behaviour and product properties. For our application something else is clearly wanted—our model is set up for the purpose of creating design solutions, i.e. to *synthesise* a solution from given premises. To achieve this we need to study and represent the human design process and all its sub-tasks. But how do we simulate human thinking processes that are ill structured, multidisciplinary and may result in not one, but many, valid answers? This is a complex task and in the same way as for all types of modelling, the challenge is to create simple models that still retain all the important characteristics of the real world.

This task is illustrated by Fig. 1.3. From given product specifications (input parameters), the design team applies its experience and product specific knowledge and comes up with a proposed solution defined by a set of design variables. The design variables should fully describe the product quantitatively as well as qualitatively and could be numerical (real, integer, sets), boolean, text strings or more complex representations. When simulating this process it is required that the same input parameters result in similar (not necessarily identical) design variables as the human designers would arrive at, representing a fully viable design solution. This implies that the flexible, dynamic and multifaceted human mind is to be emulated by a computer processor with its preprogrammed and rigid modes of operation. Clearly this is not a straightforward matter of computer programming. On the contrary, the transition from human knowledge and human working methods to computer executable instructions is the key challenge when developing design automation systems and the methods used often comes from a field of computer science called *Artificial Intelligence.*

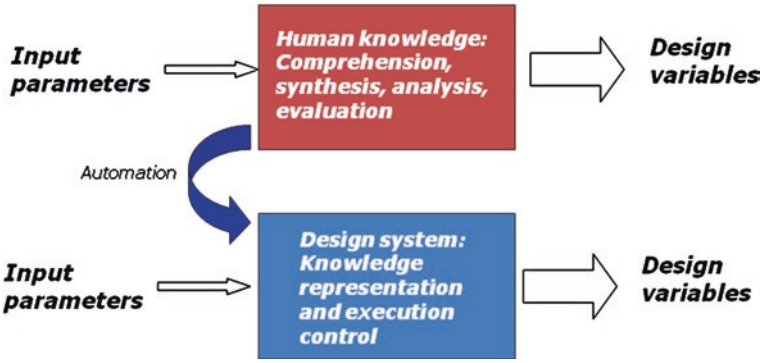


Fig. 1.3 Simulation of design work

The concept of simulating human thinking requires some comments. Similarity behind cognitive processes in humans and computers are often superficial and it is not self-evident that the aim should be to make the computer mimic human thinking. What is an efficient mode of operation for the human mind may be unsuitable or impossible for the computer and vice versa. There might well be more efficient ways to utilise the processing power of the computer—the important thing is that the results of human thinking and computer processing coincide.

It is thus important to understand the strengths and weaknesses of the computer's way of operation when applied to design tasks. It is natural to take advantage of the computer's ability to carry out long sequences of numerical operations, which the human mind is not well adapted to. Computer support enables the use of design methods not otherwise realistic, e.g. stringent and large scale optimisation processes. This is also true of long chains of logical operations where the computer vastly outperforms the human mind. Human reasoning is slow and tedious, goes on between individuals or inner monologs, but is in practice often replaced by instant response based on experience or intuition. Humans, including human engineers, do not behave as logically as we would like to think we do, making the use of computer support not only more efficient but also more reliable.

1.4 Intelligent Computer Systems

Let it be said immediately—computers have no intelligence in the normal sense of the word! They will only carry out instructions that they have been programmed to do. For many tasks such processing is entirely satisfactory but this rigid and predictable mode of operation is not what we normally associate with "intelligence". Although most people have an opinion of what intelligent behaviour is, the term intelligence itself is by no means clear-cut and a generally accepted definition is lacking. For a discussion of intelligence and computers, see e.g. (Luger 2005).

So why use the term "intelligence" at all? There is a field of computer science called *Artificial Intelligence, AI,* which focus on programming methods intended to emulate human intelligence. Many computational methods used for design automation were originally developed as general purpose AI methods. The term artificial intelligence originates from a conference held in Cambridge Massachusetts in 1956, which shows that the concept of simulating intelligence arouse very early in the history of digital computers.

What is characteristic of artificial intelligence methods is not a particular computer technology but rather the common purpose of creating computer systems to solve problems normally associated with human problem solving. This means that computers may exhibit something that resembles intelligent behaviour, while in reality the computer only processes sequences of binary numbers as is its only possible mode of operation. In principle, all computable design problems can be solved by traditional programs—the difference is in efficiency, maintainability, reliability and transparency. Programming methods close to the computer's way of

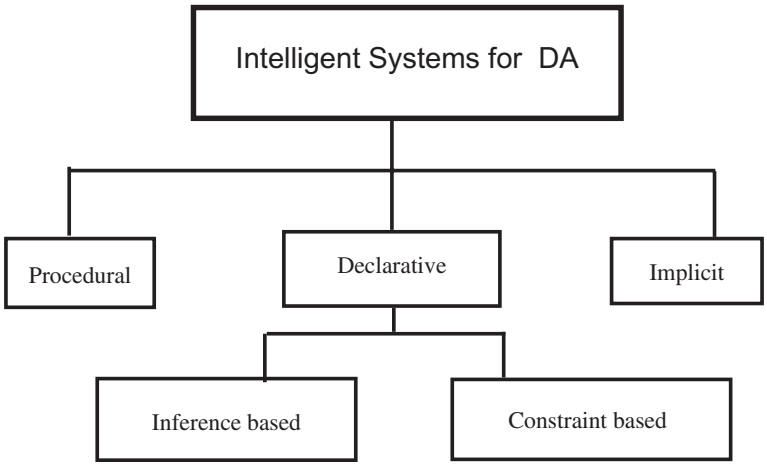


Fig. 1.4 Computer systems for design synthesis problems arranged according to the principles of underlying technology

working promote computational efficiency but are inefficient from a programmer perspective, while programming methods oriented towards user or problem representation are user friendly but less efficient computationally.

With this understanding of the term "intelligence" there is some justification in regarding computer systems used for decision support or automation as intelligent. Such systems are normally limited to restricted domains and have found applications in many fields such as speech recognition, translation, diagnosis, banking, web commerce and automatic programming.

Intelligent systems for design automation represent one such group of dedicated computer systems that perform tasks one might not expect computers to do and that use problem orientated programming methods. Two main groups of AI related programming methods can be distinguished: *Declarative Programming* and *Computational Intelligence*. A third category that should not be forgotten is the *Procedural programming* methods that represent traditional programming which also have an important place in the context of design automation. These methods often used in design automation systems are very briefly introduced below (see also Fig. 1.4) but will be given a more exhaustive treatment in Chaps. 6 and 7.

1.4.1 Declarative Representation of Knowledge or Constraints

Knowledge and constraint based systems are typically based on *explicit, declarative knowledge* and generic execution control. This means that the design rules and constraints that govern a specific problem are defined by the programmer in

a knowledge base, while the execution control rely on a generic program that will apply the relevant parts of this knowledge base as required. Hence, the programmer does not explicitly govern how the knowledge should be used, he/she only has to make sure that all elements of knowledge is correct and consistent. This division between knowledge and execution control is typical for *declarative* programming methods and reduces program size and simplifies programming and maintenance. One familiar example of declarative programs are spreadsheets where updating takes place without any direct control by the user. Two important classes of declarative programming methods often used in design automation systems will be discussed in some detail in Chap. 6: *Inference based systems and Constraint based systems.*

1.4.2 Representation of Implicit Knowledge by Computational Intelligence

Knowledge based system as well as many systems based on mathematical models use explicit knowledge in the sense that when executed they will lead directly to the required solution. For many problems it is however not practical or even possible to formulate explicit design rules. For instance, solutions might have to be found by iterative procedures that search the design space systematically. One such search based method is numerical *optimisation* that use problem specific definitions and a generic solver.

Another situation when explicit knowledge is never defined is when already existing product variants are to be used for a set of new requirements. In this case the design rules are imbedded in the existing design solutions and not explicitly formulated. This approach is known as *Case Based Reasoning*, CBR, where a special search engine scans a systematically arranged case base with existing solutions in the same product domain to find a best match.

A third example of computational intelligence methods is an interpolation method called *neural networks*. The method is inspired by the structure of biological brain tissue and consists of a network of nodes and links. The gain factors of the links are adjusted so that the entire network produces input-output results similar to a set of existing, proven solutions.

1.4.3 Procedural Programming for Mathematical Modelling

Although methods originating from the field of artificial intelligence are often used in design automation systems, it is important to remember that traditional, procedural programming methods also play an important role. For problems where the same (set of) fixed procedure(s) should always be used and where heavy numerical computations are required, the traditional programming methods

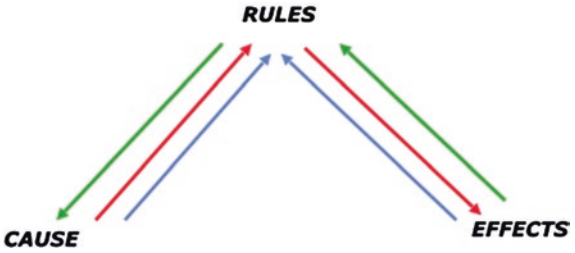


Fig. 1.5 Deductive (*red*), abductive (*green*) and inductive (*blue*) proocesses

usually turn out to be the most efficient. Having acknowledged their importance, these methods will not be given much further attention in this book since they are well known and there exist an abundance of literature for such methods.

1.4.4 The Causal Triangle

It was previously mentioned that the methods used in design automation originate from the field of Artificial Intelligence which deals with a wide range of problems. These applications adress problems with different relations between causes and effects and we need to position design automation applications from this perspective. Figure 1.5 illustrates the three principle modes of operation:

1. Initially, the **effects** are unknown. During processing, the causes create effects when the predefined rules are applied. This is a generative process called *deduction* or *inferencing*.
2. Initially, the **causes** are unknown. During processing, the causes of the effects can be determined using the predefined rules. This is a goal driven or diagnosis process called *abduction*.
3. Initially, the **rules** are unknown. During processing, the rules that relate causes to effects are derived. This is a rule derivation process called *induction*.

When we consider design automation applications the predominant mode of operation is alternative one above and the term “inference” frequently occur when discussing such systems. One could imagine also rule derivation as a possibility if design rules need to be determined as well as applications where the required goals are back-tracked to their respective sources. However, the dominating application is inferencing and this is the only causality that will be treated in this text.

References

Amen, R., Rask, I., Sunnersjö, S.: Matching Design tasks to knowledge based software tools. In: Proceedings of ASME Design Engineering Technical Conference, Las Vegas (1999)

Cederfeldt, M., Elgh, F.: Design automation in SMEs—current state, potential, need and requirements. In: Proceedings of ICED05 Conference, Melbourne, Australia (2005)

Crabtree, R., Nirmal, B., Fox, M.: Design Engineering: Problems in Coordination. In: Proceedings of JSME/ASME Workshop on Design, Tokyo (1993)
Encanação, J.L., Lindner, R., Schlechtendahl, E.G.: Computer Aided Design: Fundamentals and System Architectures. Springer-Verlag Ltd., Berlin (1990)
Luger, G.: Artificial Intelligence—Structures and Methods for Complex Problem Solving. Pearson Education Ltd, Harlow (2005)
Ullman, D.G.: The Mechanical Design Process. McGraw-Hill Book Co., Singapore (1997)
Womack, J., et al.: The Machine that Changed the World. Rawson Associates, New York (1990)

Chapter 2
Industrial Products and How They Are Developed

2.1 Man-Made Objects and the Laws of Nature

In our everyday life we are surrounded by and dependent on man-made systems and products. These are sometimes referred to as *artifacts*. We live in an artificial environment, we breathe air that comes heated and humidified out of the ventilation system, we use electronic equipment to communicate at distances out of hearing and we use transport systems to travel around the globe. When we venture a leisure trip into nature most of us bring special clothing and equipment to preserve our artificial environment as far as possible. The principles of natural and man-made objects are elaborated in depth by Nobel Laureate H Simon in his classical book *The Sciences of the Artificial*, (Simon 1996). Here we will discuss how the natural laws of physics will interact with the design rules of human origin in product development and what the implications are for the planning and functionality of a design automation system.

Natural laws are highly predictable and highly repetitive—if you throw a stone into the air, gravity will force it back to the ground with 100 % certainty. The same cannot be said for design rules created by man. The artificial laws that we impose on manmade things reflect intentions, and these intentions may well change over time. The intentional rules or constraints that we impose on a manufactured object usually reflect design knowledge based on intuition, experience or policy and are often termed *heuristic* rules. It is an important quality of a design automation system to be flexible in the sense that it allows easy change of the design rules, especially those having an artificial origin.

Of course all man-made products are subject to the inherent *natural laws* of e.g. physics or chemistry, but their properties and behaviour are also subject to rules defined by man. These *artificial laws,* represent the conscious *intentions* of the persons or organisations that designed and manufactured the products. Hence, the designer of a complex mechanical product is governed not only by constraints

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_2

originating from technical disciplines like structural mechanics, fluid dynamics, thermodynamics and so forth, but also by constraints originating from legislative or national standards, corporate policies, manufacturing system characteristics, customer specifications, or, maybe most importantly, the experiences and intuition of the individual designer. For the product to be successful neither category of constraints, natural or artificial, can be neglected.

To exemplify how these two categories of constraints interact and affect the design solution, let us consider a simple mechanical component like a threaded fastener in the form of a cylindrical steel screw. If the screw is subject to a tensional force F, has a radius r and the steel used has a maximum permissible stress of σ_y; then the required dimension, the radius, can be calculated from:

$$r = \sqrt{\frac{F}{\pi \sigma_y}}$$

Assume $F = 10{,}000$ N and $\sigma_y = 250$ MPa, then a value for r of 3.57 mm will result. A screw with a diameter 7.14 mm would thus exactly be able to carry the required load. The result follows from elementary stress theory, i.e. a law of nature.

If the fastener had been part of a living creature and made of biological tissue the diameter would probably due to the evolution of the species converge towards this optimal value in the same way as a human thigh-bone is carefully sculptured to carry the loads that it is exposed to. For a part of a man-made object however, other considerations and design rules must be taken into account. Clearly nobody would want to use a screw with an odd diameter like 7.14 mm. Such a screw cannot be bought off the shelf, it would require special stock of spare parts and no service technician would possess a socket wrench that would fit the head. The designer would instead use a standard M8 screw, thereby slightly over dimensioning the component for the sake of standardisation. Behind this decision is an intent to follow metric fastener standards and company policy.

Does this distinction between natural laws and design rules of human origin have any specific bearing on issues related to design automation? Very much so, as we will see these differences have a profound effect on how to realise such a system.

First of all the two categories of design rules and methods usually require different methods of implementation. Computer tools for representation of objects under the influence of natural laws of physics are mainly directed towards mathematical modelling and simulation. These tools thus need to have powerful functions for analytical and numerical processing. The design rules and methods that are of human origin on the other hand, are frequently of a *heuristic* nature. By heuristic knowledge in the context of engineering is usually meant simple design rules discovered through experience or common sense, see Sect. 3.2 for a more exhaustive definition.

Typically the man-made rule sets are large compilations of rather simple relations and constraints, e g lists of allowable values or combinations, sometimes

with many levels of conditional statements and nested loops. Representation of the human design intent of a product design very often requires programming paradigms that are efficient for logical reasoning and combinatorial problems. The differences between the two categories of product design knowledge demand corresponding differences in the functionality of the computer programming paradigms used in a design automation application.

There is a second significant difference illustrated by the screw example. First assume that screw diameters in the range three to ten millimetres are allowed. The proposed diameter of 7.14 mm resulting from the calculation based on elementary stress theory is a real number. The solution space of real numbers is continuous with an infinite number of solutions in the permissible range. If we introduce a design rule of typical human origin saying that only metric standard screws are allowed, then only discrete diameter values may be chosen. The solution space is then reduced dramatically and in our example consists only of M3, M4, … M10, i.e. the solution space consists of eight discrete numbers. For discrete solution variables within a limited solution space different types of *search* methods can be employed.

Limiting the solution space thus opens up for other programming paradigms than we are used to from traditional programming. For many design automation tasks computer methods based on some form of search strategy have proved very powerful. Let us return again to our example with the cylindrical screw. If we apply the search approach of *generate and test* here, we would start with the smallest screw, M3, calculate the arising stress and compare with the allowed value. If the resulting stress is below the maximum permissible value, this screw is selected, otherwise the next size is tried until a satisfactory size is found. This example is trivial and here the search based method has no real advantage over direct calculation and rounding, but for many ill structured design problems search based methods are very powerful, particularly from a computer programming perspective or may even be the only available method that works in practice. It should be emphasised again that these methods can only be used when the solution space is finite and consists of discrete solutions, which is often, but certainly not always, the case for problems governed by design rules of human origin.

2.2 A Generic Process for Product Development

The purpose of a design automation system is to simulate the product development process as it is traditionally carried out. Before we focus on design automation a brief orientation of this process is called for.

Product development is a process characterised by creativity, trial-and-error, subjective judgements, unexpected surprises and many difficult trade-off decisions. Should it be seen as a work of art or a process of science?

Management would much prefer to see product development as a scientific process. Experience however tell us that product development projects are notoriously

difficult to control. Time and cost budgets are frequently overdrawn and fulfilment of requirement specifications often becomes a matter of negotiations between the departments for engineering and marketing. In a survey of ten companies in the defence and supplier industries budgets were on average overdrawn by about 50 % and time gates by about a factor of two, (Svensson 1990). A large automotive project typically involves several thousand development engineers each with specialist knowledge of a particular part, system or property. Requirement specifications for such projects could be of the order of two thousand to three thousand measurable quantities (Sunnersjö 2003). All in all these large scale projects represent rather formidable efforts, which management cannot really control in a strict sense, but has to rely on the skill and judgement of others. Assuming that all project participants have sufficient skills for their respective tasks, the management challenge is to assure that the whole development team moves in the same direction and has a common understanding of what is to be achieved. This can be more easily achieved if there is a map of the process and this map is well known and accepted by everybody involved.

The map should take the form of a description, or model, of the product development process. In a survey (Nordström 1995) of how twelve Swedish manufacturing companies used ten popular product development methods it turned out that the only quality assurance method widely used and appreciated was when companies established a company specific product development model. The mere process of setting up such a model improves the process itself and the existence of a model clearly helps in making everybody aware of their own role in the process and how the different steps in the process depend on each other. It was generally accepted that the companies with the best development process will in the long run also produce the best new products.

The introduction of the concept of a product development model is usually attributed to Pahl and Beitz (1996). Their book, *Engineering design—a systematic approach,* first published in the German language in 1977, must surely be the most quoted publication in the engineering design literature. This book takes a significant step in moving product development towards a scientific process. It presents a detailed and logical step by step procedure for product development, parts of which was later used for the German standard, VDI-2221. Another significant contribution in the same direction was made by Hubka and Eder (1988), who introduced the concept of system theory into engineering design.

Both these works focus on engineering design tasks, which are central, but not stand-alone, activities in product development. Marketing, production planning, business activities, intellectual property rights and so forth are other aspects that must be considered. Also technical pre-studies, which are more general investigations intended to create a level of preparedness for future technical developments that may be required in new products, is part of the complete development process.

The concept of integrated product development (“total design”, “simultaneous product development”, “concurrent engineering”) became the generic panacea when planning product development processes in the early nineties with Pugh (1991) as one of the dominating text books. For most steps in the product

developing process there exist general design methods with the intention of providing stringency and effectiveness to the process. Two popular textbooks, both with a methodology focus, are Ulrich and Eppinger (2012) and Ullman (2003).

All product development models with appended methods present a simplified view of a very complex process. Nevertheless experience has proved them useful. Below an example of a simple model integrating design and manufacturing developments is presented, see Fig. 2.1. It has been used by the author for about 20 years and found acceptance where applied. It is a useful frame of reference when studying traditional development processes with the intent to set up a design automation system.

The process of Fig. 2.1 normally proceeds in a zig-zag fashion starting with the conceptual design phase generating a concept model, then moving to an inventory of production processes suitable for the planned product concept and resulting in a production process proposal. Then the process again moves back to the design leg with the preliminary design phase and so on.

In an ideal world the process will always move forward (double arrow, "synthesis") passing through all the required stages. In real life however, product development inevitably becomes an iterative process with many loops (single arrow, "analysis"). Obviously we would want these feed-back loops, indicating that something must be corrected, to be few and short. It would be very unfortunate if a serious problem, that requires a change of the chosen concept, would be discovered first when the product has been launched on the market. Much time and money would then have been wasted and even worse, customer confidence might have been upset.

To minimise the risk of late changes in the product development process it is important to analyse the proposals for design and manufacture so that unsuitable solutions are discovered at an early stage. This analysis could be performed by e.g. computer simulations, scale model testing or prototype testing. Between all stages in Fig. 2.1 there are feed-forward synthesis arrows and feed-backward analysis arrows that constitute the two elements of the iterative process.

Before proceeding we need a somewhat more stringent definition of the two terms *synthesis* and *analysis*. In this book the terms are used according to the following definitions:

- **Synthesis**: The phase where an identified engineering task or problem definition is addressed in order to find a satisfying solution based on previous knowledge and expertise (Johannesson and Persson 2004).
- **Analysis**: The phase where the product or product part solution is evaluated based on its physical representation and characteristics. Importantly, the analysis is also an activity that precedes the synthesis phase as a problem definition is broken down into manageable parts (Dieter and Schmidt 2009)

To exemplify the concepts of synthesis and analysis, consider Fig. 2.2. The truss is defined by the rod dimensions, the rod lay-out, the boundary conditions and the material properties. Using these data the resulting stiffness, K, can be calculated by e.g. the finite element method. This is an example of *analysis*.

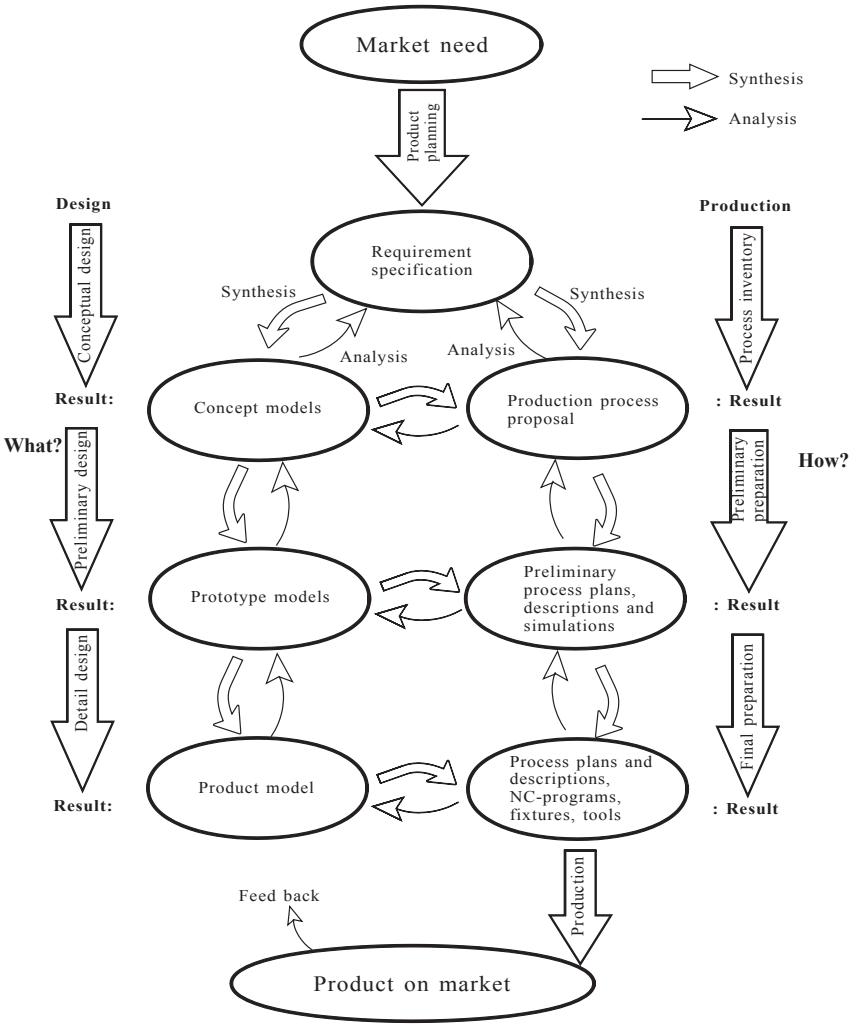


Fig. 2.1 Process model for product development that integrates product design (*left leg*) and production preparation (*right leg*). The process starts with a product plan from the marketing department and results in a complete definition of the new product and plans for its manufacture. Synthesis activities denoted by *double arrow*, analysis activities with *single arrow*. See also Rask and Sunnersjö (1998)

If we reverse the problem and instead start out with a required stiffness and length and want to define a truss structure that will provide this stiffness we have a *synthesis* problem. Clearly a large number of solutions exist, e.g. we could use thinner rods made of a stiffer material or arrange the lay-out differently. A synthesis problem is often governed by *constraints* that will reduce the number of possible solutions. For instance, it might be a requirement that all rods are of the

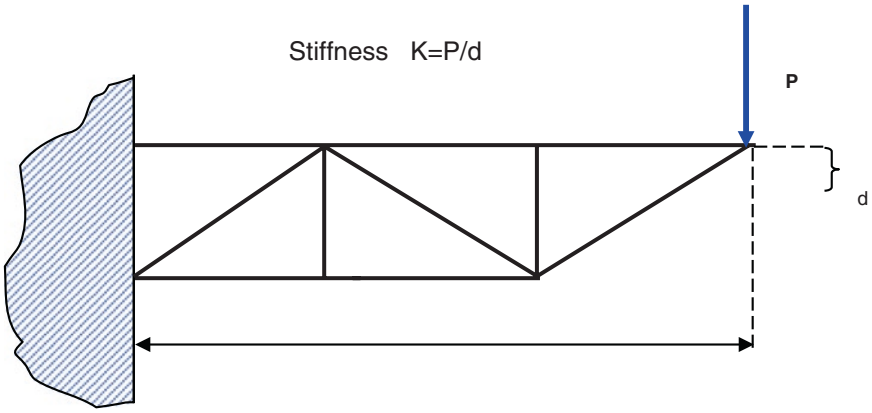


Fig. 2.2 Cantilever truss subjected to force *P* causing the displacement *d*

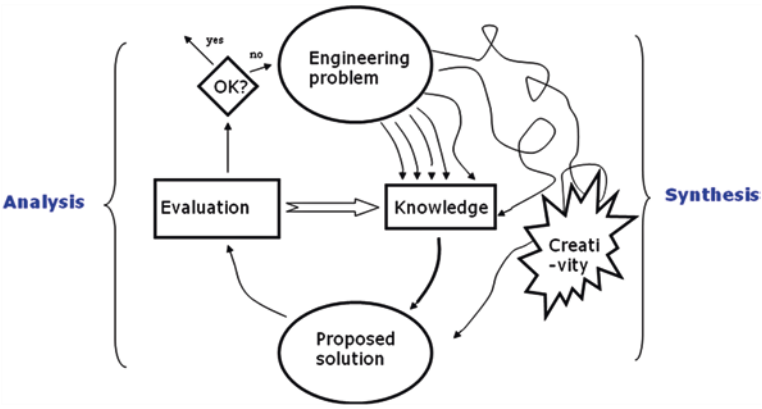


Fig. 2.3 Solving engineering problems through successive alternations between synthesis and analysis activities. See also Rask et al. (2000)

same diameter and material. If we are satisfied just by finding a solution that is consistent with the given constraints, we have a *constraint satisfaction problem*. If constraint satisfaction is not good enough, but instead the best possible solution is required, then we have defined an *optimisation* problem.

The alternation between creative synthesis actions and evaluative analysis actions is typical not only for the product development process but also for many other creative intellectual tasks. This iterative process is illustrated in Fig. 2.3. Starting from a problem definition, i.e. in our case a requirement specification for a new product, we apply our previous knowledge and experience from the specific technical field together with an element of creativity. Going from problem definition to solution proposal is a synthesis activity. As the figure illustrates the task at hand could be more or less straightforward depending on the necessity of creative, new

ideas. For design automation applications it is of course only possible to fully automate development processes where all necessary product knowledge already exists.

When the synthesis phase has resulted in a design proposal, this will be evaluated for functionality and compared with the original specification requirements. The evaluation implies analysis of the product and its functions which could be achieved by e.g. computations, expert review or prototyping. The analysis phase will result not only in a conclusion whether the specifications are fulfilled or not, but also in new knowledge about the product and its behaviour and properties. This is added to the already existing knowledge bank and improves the chance of succeeding in the next synthesis loop, should the previous one not having resulted in an acceptable solution.

2.3 Design Tasks with Potential for Automation

Over the last three to four decades amazing progress has been made on computational methods for analysis and simulation. Finite element methods, FEM, for structural and solids stress and strain analysis, Computational fluid dynamics, CFD, for flow problems, Boundary elements for potential field problems (e g acoustics) and rigid body dynamics, RBD, for mechanism analysis to name only a few. Corresponding progress in the other leg of Fig. 2.8, the synthesis tasks, has however been much slower. This is no surprise, synthesis problems are much more open, do not have a fixed solution path, have many possible solutions, are often ill structured and the knowledge base is made up by a mixture of rules of varying character and origin. Computational synthesis tasks seldom require heavy number-crunching. Instead the difficulty lies in the capture of human knowledge and human working methods within the narrow confines of the digital computer. Modern programming methods do however open up new possibilities to solve synthesis problems. The cases of applications must however be carefully chosen and this section will discuss what characterises problems that are potential candidates for automation.

2.3.1 Tasks Suitable for Automation

Selecting and defining the task(s) to automate is the first and foremost step when planning a design automation project. Figure 2.1 shows the complete product development process for original design from start to finish. It is rarely feasible to automate the process in its entirety. Instead typical design automation systems are intended for some kind of variant design and address only those parts of the process where the ratio cost/benefits is particularly favourable. It is often tempting to address too large tasks, but the key to success lies in finding appropriate limitations.

The engineering design process spans over a wide spectrum of tasks with different requirements of creativity starting with straightforward selection based on given criteria and leading to problems of original design requiring a high degree of innovation. A categorisation of design tasks according to degree of creativity was suggested by Ullman (2003) and is given below in an adapted form suited for the purpose of design automation. The list is given in ascending order of demands on creativity:

- **Selection**: Choice of individual component among predefined sets to satisfy specified constraints and objectives
- **Parametric Design**: Dimension driven geometry that adapts a predefined basic design (including topology variations) according to input specifications, formulas, methods, constraints or relations (template design)
- **Configuration**: Choice of individual components (from predefined sets) to be assembled into a system with specified properties ("catalogue design"). Choice governed by specified constraints and objectives.
- **Configuration of parametric components**: Combination of above
- **Redesign**: Includes work to adapt, modify, improve and optimise an existing design solution to fulfil new requirements
- **Original design**: The design task is defined by requirement specifications and given constraints but the principles as well as the details are left to the designer. Future variations of an original design concept belong to previous categories

These six categories of design tasks form a suitable starting point when discussing the applicability of design automation methods. Obviously we want to build such systems at low cost and at the same time reaping large benefits. How to achieve this will be elaborated more in detail in Chap. 8, but a few fundamental considerations will be discussed here. First of all the task to be automated must be chosen so that it is not mutually dependent on other tasks. If mutual dependencies with other tasks exist, these tasks will need to either be included in the same system or solved by human intervention.

One basic assumption is that all knowledge required to solve the design task is available directly or indirectly in a format that can be implemented by a computer program. A well-defined workflow for the design tasks that are to be implemented must exist or be possible to develop. These prerequisites constitute what is called a *computable design problem*. Another way of putting this, which is nearer to computer science terminology, is to say that the problem must be possible to solve subject to *the closed world assumption*. The implication of this assumption is that all facts are regarded as false unless they have a value that is either stored in the computer memory or possible to infer.

These criteria indicate that products based on mature, proven and well documented technology are good candidates for automation. The more stringent and explicit this product knowledge is, the easier it is to program it into executable code. Referring to the list above it could be concluded that the first four design categories, *selection, parametric design, configuration, configuration of parametric components*, are the primary candidates for automation. Original design is not

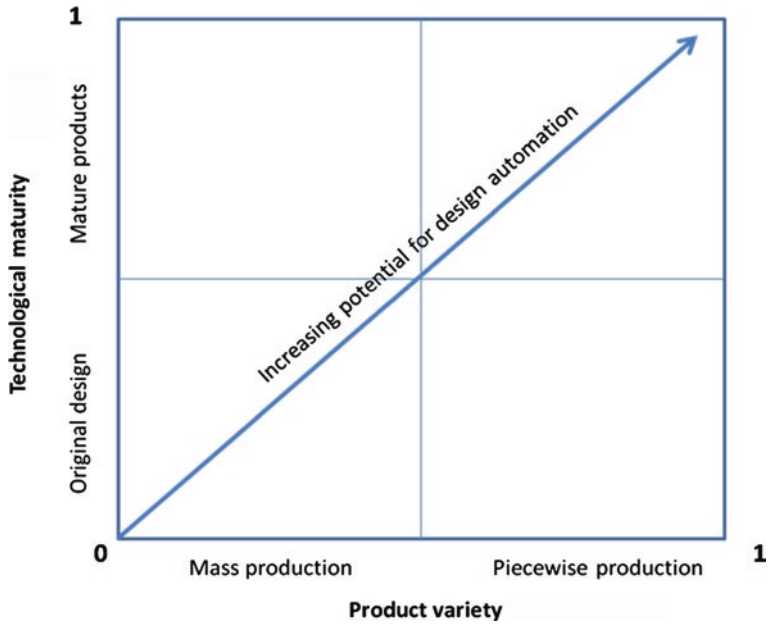


Fig. 2.4 The potential for design automation is high for products being technologically mature and being produced in many variants. The value 1 on the *abscissa axis* denotes that all products delivered are unique, while the value 1 on the ordinate axis denotes that all the required design rules are known and documented

a suitable application, while redesign offer some possibilities for more advanced forms of design automation.

The previous discussion concerned the feasibility or ease of building a design automation system for a given task. The other consideration when selecting tasks to be automated is what benefits that could be expected. Often customisation has a value in its own right and is the dominating driving force. Providing fast and accurate quotations for customer adapted products is a specific business factor where design automation might be crucial. Generally speaking, it is when a product will need to be redesigned in a large number of variants and when design cost or design lead time become a problem that it will be advantageous to use design automation methods. Figure 2.4 illustrates how the potential for design automation increases with product variety and technological maturity.

The previous discussion implies that variants are sold directly to the customer and that the ability to meet requirements of products adapted to customer requirements have a significant business value. There are however also useful applications of design automation where many variants are created for the internal development process but only the final version is actually sold to the customer. This kind of *iterative* design process often has the objective of optimising certain properties of the product and the design automation system becomes an instrument for this process.

2.3.2 Design for Variety, DFV

The process of product development will be different if detail design and manufacture is initiated by a customer order compared to when the product is designed, manufactured and stocked, waiting for a customer. The two alternatives are illustrated in Fig. 2.5. The first three steps are similar, but while the traditional, market based process continues until the product is launched on the market, the order based design makes a halt until a customer order with specifications is placed. Only then is detail design and manufacture carried out. The traditional market based process means that the marketing department takes the role of a customer

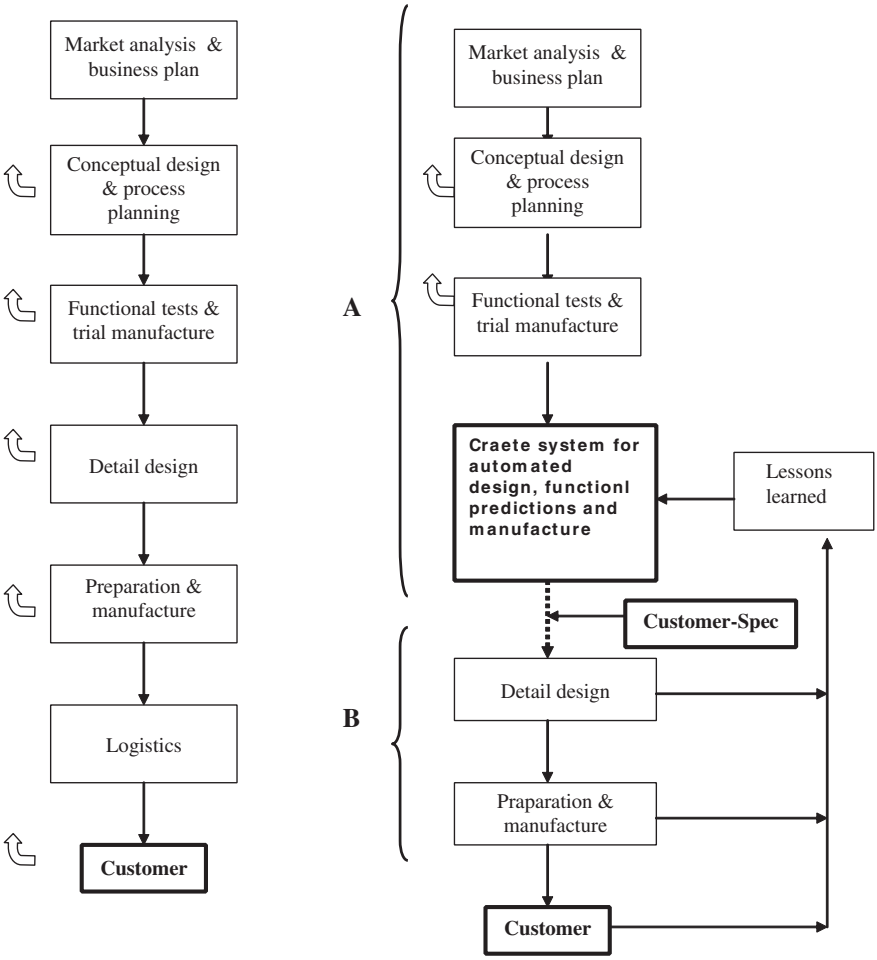


Fig. 2.5 Market based and order based design and manufacturing processes. Part A is carried out prior to order, part B when an order has been placed

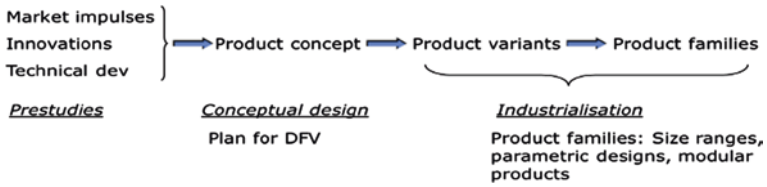


Fig. 2.6 Design for variety: from pre-studies to product families

representative and specifies the product trying to anticipate what the market preferences will be at the time the product is launched. Needless to say, considerable resources have been spent before the product meets the customer and if development time amounts to several years, there is a risk that the internal specification has not been on target.

The order based process avoids much of these problems but requires some sort of automated design system and a manufacturing facility with high flexibility. For products that are built in short series and high costs, e.g. ships, power plants, railway cars, production equipment etc., order based design is the self-evident way of working. But there are many degrees of variations available where the product platform is market based and the equipment is order based, e.g. in the motor car industry.

Standardisation is a core value in almost all efficient industrial processes. All experience indicate that industrial success is promoted by standardised products and procedures. However, as we have seen, with increased global business environment there is also a trend towards increased product variety. How can these contradictory trends be reconciled? How can flexibility and standardisation be combined? Design automation methods have an important role in this respect, but of course, the product itself must be designed in such a way that the required variations can be realised with relative ease. The methodology to achieve this is called *Design for Variety, DFV,* and the corresponding industrial process is illustrated in Fig. 2.6.

From a business point of view one can distinguish between four categories of design variations.

- **Select from stock**: All available variants are designed and manufactured prior to customer order
- **Configure to order**: Standardised modules and components are assembled in a variable product structure
- **Modify to order**: Use of a generic product structure with adaptable modules and components
- **Engineer to order**: Use variable product structure and adaptable modules and components

Customer influence and customer involvement in the product development process is low at the top of the list and increases as we move downwards. For *engineer*

to order contracts, e g in shipbuilding, power plants and production equipment, the customer usually plays a very active role and often defines requirements and choice of technology. At the top of the list, it is up to the manufacturer to predict what preferences the potential customers will have when the product is ready to launch. The latter situation, e.g. for production of motor cars or house-hold appliances, involves a higher risk because much has been invested in the new product before it is launched and the market response can be evaluated.

Irrespective of what business case is at hand, in order to be successful in DFV it is important to plan for *product families* and not for singular products from the beginning. By "product family" we mean a group of related products that to a high extent share the same features, components or modules. A viable product family concept will consist of products where a large proportion of the products have a common set of features, components or modules that remain the same within the whole product family. This common set is called the *product platform.* If a high proportion of the products are made up of this platform the product family is said to have a high degree of *commonality*. If standardised features, components or modules are used in different platforms these are said to *carry-over*. Of course, methods like commonality and carry-over save resources and should be exploited when possible without sacrificing the variety that the market demands.

To satisfy the requirements of a DFV process, the "Product planning" phase of Fig. 2.1 should thus result in a specification for both the product platform and the complete product family, i.e. the full range of planned products. It will be much more difficult to expand this range at a later stage. A good rule is to always aim for exploration of the principles of standardisation when planning the product range.

2.4 Investing in New Products

Development of complex products is a costly process with inherent risks. For a manufacturing company however, updating of its product range is a necessity if the company hopes to survive in the longer perspective. To give an indication of the economic aspects of product development some examples from the motor car industry will be given below.

A rule of thumb for the relative development costs attributes about 2/3 of the total budget to product development, i.e. the left leg of Fig. 2.1, and 1/3 to what is often referred to as the *industrialisation* phase, which is the right leg plus acquisition costs for tooling, production equipment, premises and so forth.

For the total project development costs some examples from the history of the automotive industry are given in Fig. 2.7. Note that the vertical axis is graded in a logarithmic scale, which means that the increase in development costs, which appears to be linear with time, in reality is exponential. The trend points towards twofold cost increase over 15 years and in 1995, when the graph was made, development costs accounted for between 5 and 10 % of the turnover for typical mass producing automotive manufacturers.

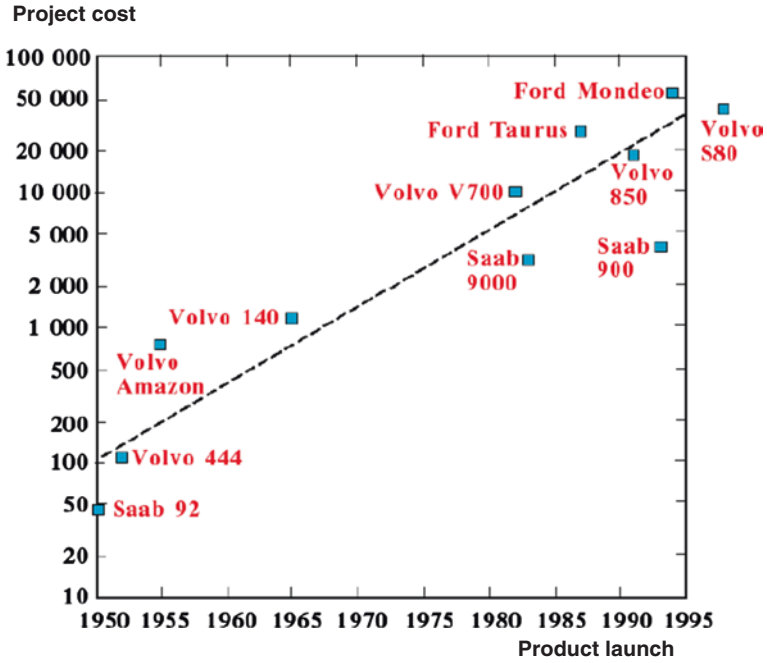


Fig. 2.7 Total project costs for car models launched over 50 years. Costs in million SEK (one SEK approx. 9 euros), compensated for inflation and normalised to 1995. Numbers are rough estimates based on information reported in news media and interviews with development personnel

Although the costs given are rough estimates based on indirect information and some rather crude assumptions, the general trend of costs escalating out of proportion is indisputable. This is of course a situation that cannot be sustained in the longer perspective and the industry has also applied various approaches to arrest the accelerating development costs, which at this time seemed to be out of control. Apart from a general strive to improve efficiency in all phases, most methods boil down to an effort to distribute the heavy development costs over as many units as possible.

Popular concepts like *outsourcing*, *commonality* and *carry overs* have proved their worth to control cost escalation during the last two decades. The methods all have in common that they, in different business scenarios, seek to reuse existing, proven designs, adapt them to new specifications and avoid costly tasks of original design starting from scratch.

Take outsourcing as an example. This approach implies that the car manufacturer abstains from the responsibility of developing new technology and new designs for those parts of the car that are not considered to contribute to the *core value* of the brand. Instead a supplier takes this role and specialises in certain components. This supplier will usually have several industrial customers who want much the same

component, but with various specified adaptions. The core technology is the same and many parts thus share the technical development costs. It is then up to the supplier to be able to adapt the design correctly, quickly and cheaply so that each customer specification is satisfied and, recognising the archetypes of variant design as described in Sect. 2.3.1, this is where DFV by design automation has an important role to play.

Commonality is the term used where many models in a *product family* to a large extent share the same components and *carry-over* is when parts and modules from previous designs are reused in new projects. In these three efforts to take control over accelerating development costs, the concept of design automation has the role of enabling technology. Adaption and reuse of proven design solutions saves engineering time, but this is not the main saving. If proven solutions can be used the heavy costs of prototyping, testing and regulatory approvals can often be avoided. Combining reuse of proven technology with state of the art design automation systems is a very competitive concept.

References

Dieter, G., Schmidt, L.: Engineering Design. McGraw-Hill, New York (2009)
Hubka, V., Eder, W.: Theory of Technical Systems. Springer, Berlin (1988)
Johannesson, H., Persson, J.-G., Pettersson, D.: Produktutveckling (Product Development). Liber, Stockholm (2004)
Nordström, L.: Kvalitetsmetoder i konstruktionsarbetet (Quality Methods in Engineering Design), vol. 1. Swedish Association of Manufacturing Industry, Stockholm, Sweden. Faktarapport No 1103-7067. (1995)
Pahl, G., Beitz, W.: Engineering Design—A Systematic Approach. Springer, London (1996)
Pugh, S.: Total Design. Addison-Wesley Publishers, UK (1991)
Rask, I., Sunnersjö, S.M., Amen, R.: Kunskapshanterande IT-system för produktframtagning (Knowledge based IT systems for Product Realisation). IVF Publication No 00823, Gothenburgh, Sweden (2000)
Rask, I., Sunnersjö, S.: Konceptkonstruktion—Val av material och tillverkningsmetod (Conceptual Design). IVF, Göteborg (1998). ISBN 91-89158-08-3
Simon, H.: The Sciences of the Artificial. MIT Press, London (1996)
Sunnersjö, S., Rask, I., Amen, R.: Requirement-driven design processes with integrated knowledge structures. In: Proceedings of DETC03, ASME 2003/CIE-48218, USA (2003)
Svensson, P.: Management of product development. PhD thesis, Chalmers university, ISBN: 91-7032-479-4 (1990)
Ullman, D.: The Mechanical Design Process. McGraw-Hill, New York (2003)
Ulrich, K., Eppinger, S.: Product Design and Development. McGraw-Hill, New York (2012)

Chapter 3
Computerised Methods to Design for Variety, DFV

3.1 What Are the Deliverables of the Design Process?

In the previous chapter we discussed which design tasks that might be candidates for design automation from a technical feasibility perspective but also how a sound business case for such automation could be identified. An industrial company would obviously want to first identify its position in the business related framework (called DFV above) and verify that an automation effort will pay off its investment costs. It would then proceed to analyse if and how this business process could be realised with available technical means. The product definition is the final deliverable from the automation system, hence we now need to consider what such a system is required to produce. Just as the human designer uses IT-tools like spread-sheets, CAD (Computer Aided Design) and PDM (Product Data Management) systems to define the designed product, these tools will also form parts that are more or less integrated in an automation system. Instead of being operated by human designers they will interact with a computerised knowledge processor.

The end result of the design process, whether it is manual or automated, will be a *product definition* or a *product model*, that is sufficiently complete for the production department to acquire materials and components, manufacture parts and then to assemble the complete product. In this context the product model will be expressed in terms of its *design variables* that completely define the product.

However, the industrial process neither begins with nor stops with the product design. A large number of other tasks need to be carried out before the product is ordered, designed, manufactured, delivered and paid for. In an ideal world the product model would be a complete definition of all product knowledge needed

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_3

for the entire industrial process. Apart from purchasing, manufacturing and assembly information, it would also contain information for:

- Functional description with requirement specifications,
- Time and cost estimates for quotations and invoicing
- Technical calculations and simulations,
- Manufacturing preparation. Example: Programming of paths for numerically controlled machining,
- Control of geometry, finish and properties. Example: Measurement tables and inspection and testing instructions, programming of numerically controlled measurement machine
- Products in different phases of production. Example: Geometry of blanks and preformed parts
- Design of tooling, fixtures and jigs,
- Planning and monitoring of deliveries,
- Publishing of documentation and manuals,
- Life cycle instructions, i.e. how to distribute, maintain, disassemble and recycle the product,
- Immaterial property rights such as acquisition of licenses for registered patents
- Documentation of the design rules and constraints that have created the product.

To set up such an exhaustive and complete product model as described above would be very cost and time consuming and it is unlikely that it could be implemented as a unified, computerised model. A complete product model in this sense might not be worthwhile or even technically feasible, but by starting with the automated system directly related to the design of the product it is later often possible to find highly profitable additions for downstream tasks in the industrial process. Quotations for instance, is at the heart of the business process and by automating cost estimates based on preliminary designs, the quotation process can often be improved dramatically both regarding lead times and accuracy. The same can be said for production preparation and control tasks. Making design rules available to production could help to improve manufacturability without violating functional requirements. Having integrated product, production and control data it often turns out that drawings could be simplified or even become superfluous and hence, the decision to integrate and automate the process will in itself affect what needs to be automated.

Product design is the natural starting point when building an automation system, but very substantial benefits in other parts of the industrial process can often be found and should be exploited. Having emphasised the importance of not restricting the attention to product design, in this text we will continue to concentrate on this specific task in order to maintain our focus. The methods used in design automation are often equally applicable in other tasks in the industrial process.

The content and character of the product definition resulting from the design process will of course vary from case to case, but for mechanical engineering products one can usually distinguish three main types of information:

- Product geometry, e.g. drawings, CAD models
- product structure, e.g. Bill Of Materials, Product Variant Master
- technical specifications, e.g. heat treatment, welding specs

We will now discuss these information categories in more detail. For each category a short frame of reference for the respective design tasks will be given and the conclusions for automation of these processes are discussed.

3.1.1 Definition of Product Geometry

Product design and documentation is today almost entirely done using CAD systems. These commercial systems used for geometry generation and storage are very large software packages that build on sophisticated technology. In this section the origins of these technologies are reviewed together with some basic principles and the implications these principles might have in the context of design automation.

2D CAD
The most common documentation of an engineering product is the two-dimensional drawing. The traditional drawing board is today replaced by two dimensional CAD systems, which increases productivity but in principal does not change the working methods or the product definition documents.

An artist's impression of an object is a free illustration of what the artist has perceived. A technical drawing could also serve as an illustration but is something fundamentally different. A 2D drawing is a representation of a 3D object using standardised projections and sections. To produce clear and correct drawings require considerable training and experience. Both the producer and the user of the drawing must understand and follow the same standard, otherwise the product definition might be misunderstood. A 2D drawing is not in itself a complete definition, instead it must be interpreted by a knowledgable person. Also for an experienced designer it can however be quite difficult to understand a 2D drawing of an unfamiliar object and to translate it correctly into a physical 3D object. Generally speaking, 2D drawings are not computer readable which is a clear disadvantage in the context of design automation. For certain simple geometries that genuinely are in 2D, like bodies of revolution, a parametric 2D CAD system is fully satisfactory as the geometry module in a design automation system, but for more general products, something else is required.

3D CAD
Many of the shortcomings of 2D drawings can be eliminated by using three dimensional representations. This means that the complete model extends into space

having three dimensions, while the elements that make up the model could be of dimension zero (points), one (straight lines, curves), two (surfaces) or three (solids).

There are in principal four types of three dimensional CAD models: *Wireframe models, surface models, volume models and solid models*, as illustrated in Fig. 3.1. *Wireframe models* consist of elements of dimension zero or one, which define vertices and edges of a three dimensional object. Wireframe models have limited ability to represent free form three dimensional objects, they do not fully define the object and for more complex products they become very cluttered. As computer capacity has increased more advanced representations have consequently emerged. The first step is to cover parts of the wireframe model with flat or sculptured surfaces, thus extending the dimensionality of the allowable geometrical elements to two, which then form a *surface model*. If all wires are covered with surfaces and these surfaces form a connected shell without openings or slits, this is called a *volume model.*

The volume model appears similar to the real object but nevertheless is an incomplete representation that needs to be interpreted. The model shows the product geometry but lacks the information of how it is put together which, as we will see later, is necessary when the model is to be parametrically controlled. A three dimensional model that fully defines the geometry of the product consists of vertices, edges, directed surfaces ("directed" to define on which side the material is) and a topological description of how the elements are connected. This is called a *solid model* and has become the dominating geometrical representation in advanced design in mechanical engineering. Since the solid model is complete it can be used to draw required information for other tasks in the design process, e.g. creation of drawing documents, analyses and simulation, property calculations, calculation of NC tool paths, interference control, visualisation and so on. The solid model can be made parametric, see Sect. 3.2, and then becomes a very efficient tool to handle geometry in a design automation system (Fig. 3.1).

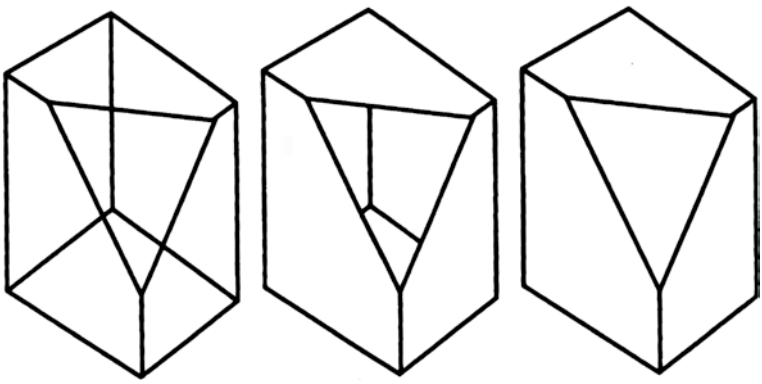


Fig. 3.1 Four types of CAD models in 3D: Wireframe (*left*), surface (*middle*) and volume or solid models (*right*). Note that volume and solid models appear similar. See also Amen and Sunnersjö (1996)

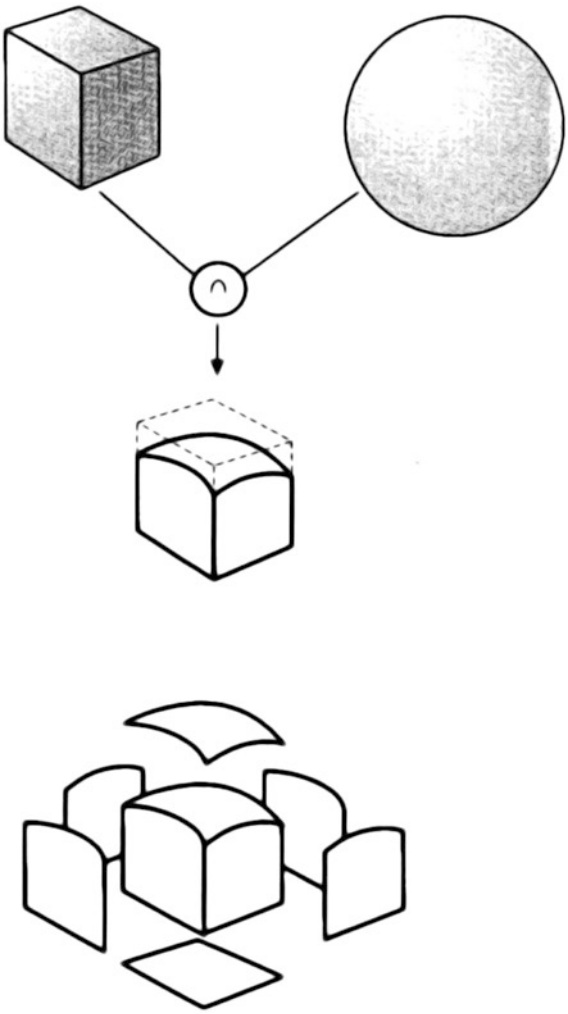


Fig. 3.2 Building a solid CAD model (*middle*) by CSG (*top*) or B-rep (*bottom*) methods. See also Amen and Sunnersjö (1996)

Solid CAD models are created and stored along either of two main principles or combinations and developments of these: *Constructive Solid Geometry, CSG*, or *Boundary Representation, B-rep*. The two principles are illustrated in Fig. 3.2.

The CSG method uses a library of predefined, adjustable building blocks, e.g. cube, sphere, cylinder, wedge and so forth to build the intended geometry by boolean operations like "A union B" (corresponds to: Join objects A and B, e.g. by welding) or "A difference B" (corresponds to: Remove material defined by object B from object A, e.g. by machining).

Obviously the standard building blocks, called *primitives*, cannot create any complex geometries. The primitives can therefore be complemented by user defined building blocks, or *features*, where a 2D profile is defined and transformed

into a 3D object by extrusion, revolution or following some more complex trajectory in the third dimension. The 2D profile can be made dimension driven and is updated using either a parametric or variometric method. The former is based on execution of the stored chain of operations that created the basic profile, while the latter method is based on solving the system of equations that define the profile geometry mathematically.

Some CAD systems have predefined building blocks for common manufacturing features, *form features*, like fillets, rounds, counter-sunk holes, dove-tails and so forth in addition to the fundamental primitives. Building geometry models by adding stepwise more detailed form features reminds of machining a blank to net shape and is a work method that seems to appear natural to engineering designers.

In CSG models not only the resulting geometry is saved but also the construction history of this geometry. This *history tree* is a kind of recipe for the solid and provides an opportunity to create product variants, which is a method of great importance in applications for design automation as we will see later.

Solid CAD models can also be built using the B-rep method. This is a more direct method that is used for objects with complex geometrical shapes and free form surfaces. The solids are defined by their vertices, edges and directed surfaces and this is the only information stored. There is consequently no history tree, which means that the model cannot easily be changed and also that the model's correctness depends on the accuracy with which it has been defined. The latter property often causes problem when transferring models between different CAD systems working with different accuracies, which might introduce erroneous slits or overlaps between surfaces in the transferred model.

In practice most commercial CAD systems for solid modelling use a combination of CSG and B-rep methods. In such systems a B-rep model that is complete and without any slits or other defects is automatically transformed into a solid model that can then be further developed by use of boolean operations either with primitives or other free form solids.

More advanced company confidential technology for modelling with and without history tree is implemented in the large commercial CAD systems and when using these as part of a DA system it is important to test how model updating actually performs in practice in the specific system. Generally speaking a parametric solid modeller is the natural choice of geometry tool for design automation applications and further on we will discuss how to use such systems for parametric design.

In the discussion above we have taken for granted that a model defining the geometry of the product will always be required. This is however not necessarily the case and if this is not so, we should not waste resources to produce such documents. It is common practice for some products that the tools rather than the actual products are being modelled in the CAD system. It might appear surprising that no product documentation exists, but for some products formed by e.g. pressing and deep drawing, the tool geometry is much more important than the details of the pressed object, since the tool geometry is required to program the machining of the dies. This is exemplified by the industrial DA system described in Appendix A.5.

2D drawings are often the end result of a design process, one reason being that legal documents in this form are often required, e.g. as a part of a contract with a sub-supplier. There are however many opportunities to omit drawings altogether and use the digital 3D solid model as the master representation of the product. Sometimes it might be possible to go one step further and store only some sort of "recipe" of the product model, which is then used to regenerate the geometry when required. In the industrial DA system described in Appendix A.3 only the input parameters to the automation system are stored while the actual product definition is discarded once the product is delivered. Should any particular drawing be needed later on, the system is executed with the new parameters.

When planning a design automation system all these opportunities to simplify the design process and reduce the amount of documentation, whether digital or on paper, should be considered and exploited when possible. As pointed out before, a design automation project is a golden opportunity to review and streamline all aspects of the design process. The buzzword "Lean", in the sense of removing all non-value adding operations, is highly relevant in this context.

3.1.2 Definition of Product Structure and Technical Specifications

A company delivering complete products to the end customer is called an OEM (original equipment manufacturer). The complete product is assembled from individual parts and modules, which could be manufactured in-house or bought from outside suppliers. Although the OEM has the full responsibility for the product, the insight into some of its constituents might be limited if the product is complex and contains many bought parts.

On a traditional assembly drawing the identifiers for components and part drawings are defined by a written list, a "bill of materials", BOM. In a solid CAD assembly the corresponding information is given as an underlying *product structure*, which is a directed tree graph, see Chap. 5, defining what components make up the product and in which order they are assembled. The assembly tree could have leaves (parts) or branches (modules) that can be chosen from several alternatives by the customer, thus opening up possibilities to customise the product for the individual buyer.

The tree structure can be represented directly in the CAD system or a dedicated archive system, but a more versatile tool is a PDM (product data management) or a PLM (product life management) system. Such systems allow controlled sharing of information, authorisation procedures, safe file storage and the establishment of workflows for frequently recurrent tasks.

The components could be tailored and manufactured for one specific product, but many components are standard parts bought off-the-shelf and where the OEM possesses no detailed knowledge or documentation. Examples of such components are fasteners, ball bearings, hydraulic actuators or electronics. Geometry is

represented as outside dimensions, while the inner properties are defined by some sort of *technical specification,* e.g. for a ball bearing stating type, main dimensions, maximum speed and maximum static and dynamic load.

Technical specifications are also required for manufactured components regarding materials and specific details of the manufacturing process not apparent from the CAD model. This could be heat or surface treatment, but also specific control or test procedures that the component is required to pass. Further, the technical specifications could refer to company or legislative standards that the product is required to follow.

3.2 Parametric Design

Many products are designed in a series of sizes, i.e. the *basic geometry* is varied in steps to create a *size range* from which a customer could select a suitable variant. Examples of products that are usually planned and designed in a size range are standard catalogue components like bearings, belt transmissions, valves and so on. Products in this category are normally manufactured and stocked until the component is *selected* by the customer. Alternatively, the size range could be "virtual" and manufacture does not start until an order is placed and the basic design is then *modified* according to customer specifications. The degree of customisation is naturally higher for modified rather than selected products. Usually some main dimensions are varied in steps to cover a suitable range. For some products where "form gives function", e.g. a fan blade, the product is scaled with proportions maintained, while other products might well change both in size and proportions. For some components it might by more natural that some non-geometrical entity is varied, e.g. the power of an electrical motor. The required steps in these entities will then need to be transformed to corresponding geometrical dimensions, often by use of domain dependent scaling laws.

A parametric design is governed by a number of driving dimensions., which could be a small fraction of all dimensions required to define the product. The driving dimensions should ideally be independent of each other and govern the remaining driven dimensions through constraints, usually of a geometrical (e.g. "Perpendicular to") or mathematical (e.g. "Area = length * width") nature. The causalities between driving and driven dimensions are established when the model is built and cannot easily be changed. Defining which are the driving dimensions must thus be carefully planned. In this book the driving parameters, whether geometric or not, will be called input *parameters* while all the resulting output variables will be referred to as *design variables*.

The traditional way of defining the geometry of parametric products is by a *template drawing*, consisting of a schematic drawing of the basic geometry with symbols for the driving dimensions together with a table giving the numerical value(s) of these dimensions for a specific product family. With the arrival of parametric 2D CAD systems, the template drawings have been replaced by 2D

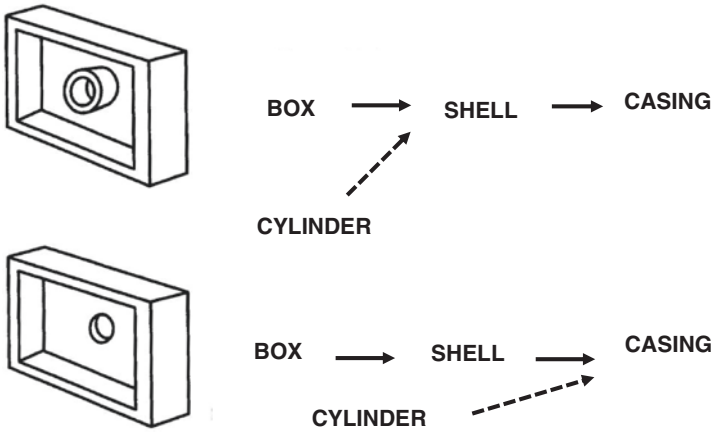


Fig. 3.3 Solid model resulting from the same primitives and the same Boolean operations but executed in different orders. *Dashed arrow* indicates operation "CUT". Adapted from Amen and Sunnersjö (1996)

drawings that adapt the product geometry to the specified driving parameters so that it is correct in scale and proportion.

Owing to the CSG building method and in particular the stored history tree, the solid model could be made parametric in much the same way: The driving dimensions are given as input parameters, the driven dimensions are evaluated and the operations defined in the history tree are activated step by step using the updated dimensions. A solid model will result, that is correct in scale and proportions. Note that the order of operations is decisive for the resulting solid as exemplified in Fig. 3.3.

Except for adapting the basic geometry to new dimensions, a solid model could also undergo *topological* changes, see Fig. 3.4. This is achieved by including optional elements (features) in the history tree with flags "on" or "off". These optional elements could be user defined or taken from a more general library of parametric *form features* that relate to the relevant manufacturing methods or it could be *design features* that make up the functional elements of the component.

This way of building a complete solid model from parametric elements, like primitives, form features, design features and more complex user defined features is an efficient way of working with solid modelling. The real advantage comes however when the model is built to allow automatic adaption of driving dimensions and topological variations. This is possible only if the parameters of the parts used are linked together by constraints so that if one parameter is changed, this change will propagate through the model updating all dependent parameters, thereby maintaining its basic geometrical properties.

These properties are sometimes self-evident, a threaded hole intended for attaching a screw should, with few exceptions, be perpendicular to the surface, a smooth transition between two surfaces is achieved by a tangency constraint and

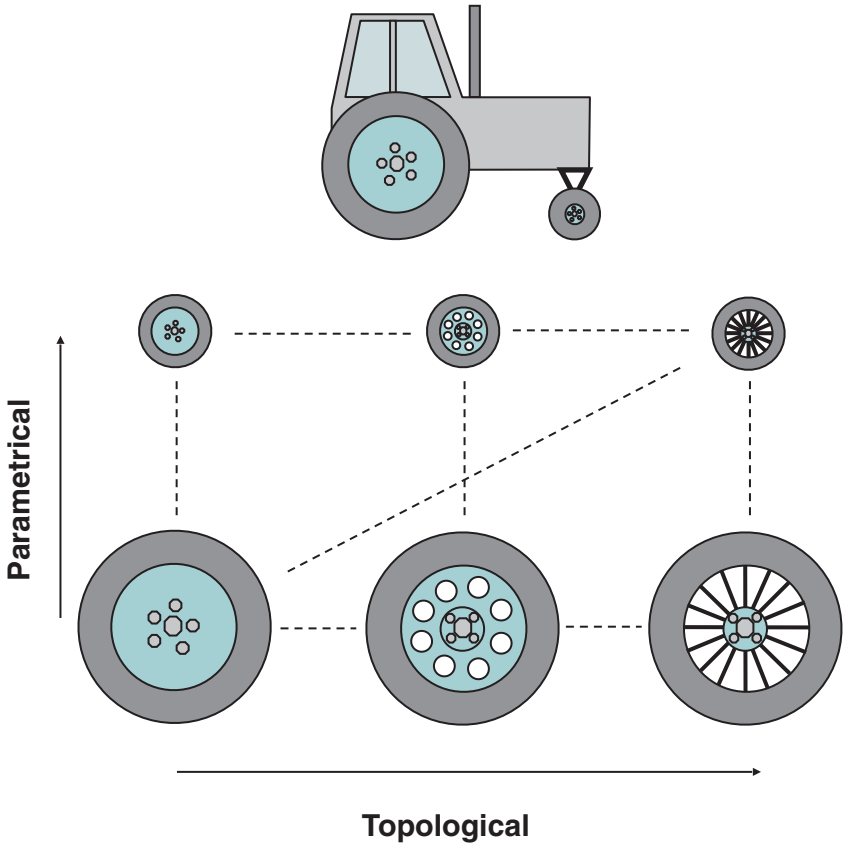


Fig. 3.4 Product range of tractor wheels with dimensional and topological variations. From Elgh (2009)

so forth. Other cases might be more complex and require more elaborate constraints to retain the essential geometrical characteristics after updating.

The constraints represent some kind of intent that the designer has for the product geometry and this intent should be preserved also when the geometry is updated with a new set of driving dimensions. Often this intent is clear enough but there might well arise situations where a parameter change also alters the topology of the component which might be acceptable or not. This is something the CAD-system cannot decide—it is a situation where the designer must intervene. A simple example is given in Fig. 3.5.

Generally speaking, to construct complex, parametric solid models that allow parametric changes of both topology and dimensions can be a challenging task. There are examples of models with several hundred driven dimensions and needless to say, the likelihood of such a complicated model to update reliably is small.

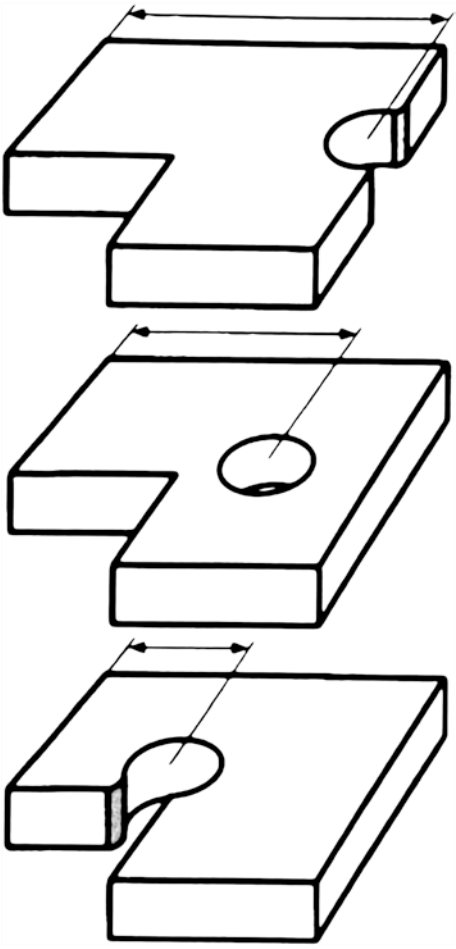


Fig. 3.5 As the hole is moved from right to left a new corner is created and will be automatically rounded. Is this what the designer intended? From Amen and Sunnersjö (1996)

If the model is planned to be a part of a design automation system and be used frequently over many years it requires careful planning to assure its functionality. Some things are obvious like not using a feature that could be turned "off" as a reference for any dimensions, or define restriction of the ranges of the driving dimensions so that the model does not collapse. Other things are difficult to foresee and exhaustive testing of the stability of the model is necessary.

An important first step when setting up a parametric model is to define what dimensions should be the driving dimensions (input parameters) and what dimensions should be the driven dimensions. Together driving and driven dimensions will be the list of design variables that define the product or component. The input parameters must be chosen so that all main dimensions that might be required to be changed can be controlled. Subsequent to updating, these will then generate updated driven dimensions. This causality among the design variables are

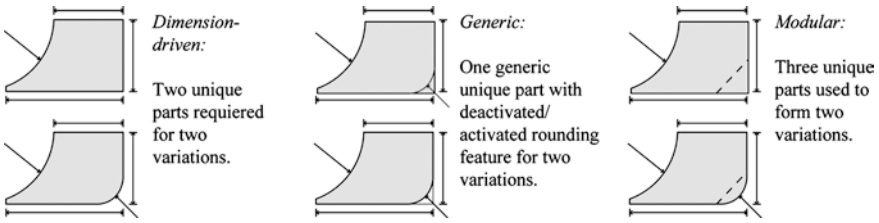


Fig. 3.6 Three methods to create parametric geometry. From Cederfeldt (2007)

important because often it is not possible to change the defined order later without having to rebuild the model.

There is an element of *reversed engineering* in the process of determining the relations between driving and driven dimensions. The similarity is due to the fact that a basic design for the intended range of variations normally exists as a starting point. To build a product size range one must then for each driven design variable clarify the origin of that variable and how its value has been determined, e.g. by deriving relations to the driving dimensions or directly to the requirement specification. Such an analysis process provide an excellent opportunity to scrutinise and improve existing design practices. The concept of driving and driven dimensions and their relations is further discussed in Sect. 8.3.1 and illustrated by Figs. 8.4–8.6.

Three approaches to model parametric solids will be discussed below and these are shown in Fig. 3.6. The example chosen is a model of a plate that allows change of four dimensions, (three of which are independent) and two topologies (corner rounded or not).

The *dimension driven part* method simply means that two dimension driven parts, one for each intended topology, are created. This method is obviously very reliable, but for complex geometries and many combinations it will require many parts and many of the design rules used will have to be duplicated. If however the parts to a large extent are standard and are often reused for the products they are intended to, it might well be worthwhile to create a library of such parts.

The *generic part* method handles topology changes by activating or deactivating the required features. For our example only one part is thus required. The topology variation parameters are represented in a similar way as the geometrical parameters which is advantageous for the clarity of the model. For more complex geometries however, the relationships between different parameters become more complicated to handle and also there is a risk that some parameters might have relations that are topology dependent. Although efficient there is thus a significant risk for malfunctioning that needs to carefully investigated when using this method to build a parametric model.

Finally, the *modular part* method uses small, stored parts that can be assembled into all the required topology variations. The parts are parametric, stored in a library and combined using Boolean operations into the required topology. For our

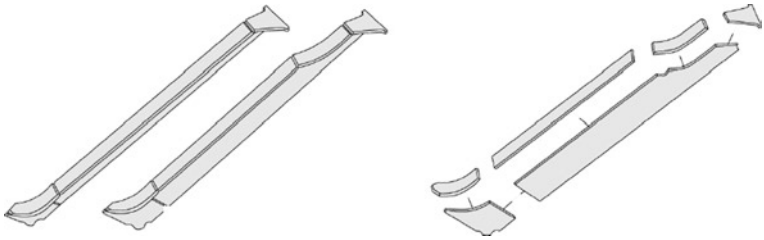


Fig. 3.7 Two variations of steel stiffener for a ship bulkhead (*left*) and the parametric parts of which the stiffener is constructed (*right*). From Cederfeldt (2007)

example three parts are required. This disadvantage of needing three parts is balanced by not having to program the model to activate/deactivate any features.

The three methods described will often need to be combined in practical applications. The stiffener shown in Fig. 3.7 is constructed by a combination of the dimension driven part method and the generic part method. The model has been used extensively and proved stable and flexible. When working with parametric components there is always a risk of successively overloading the model with too many rules and constraints. What might originally have been a robust parametric model might become too complex as new design rules are encountered and implemented. The complexity might cause loss of overview and unforeseen problems with updating. One solution might be to store parts of the design rules externally. The balance between knowledge built into the CAD model or stored and processed outside is discussed further in Chap. 8.

3.3 Configuration Design

Imagine sitting in an airplane descending towards O'Hare, Shiphol or some other major international airport. The airports are surrounded by thousands of parked cars and the cars within sight represent a tremendous effort of technology, manpower and organisation. Despite the large number of cars it is unlikely that any two cars are identical—most modern cars are built individually according to the manufacturing paradigm of *mass customisation*.

As you bring your car out of the parking lot the approximately 30,000 parts of which it is assembled work together both technically and aesthetically. When the car was built each part was delivered on order based demand by an intricate network of suppliers and has appeared at the right moment at the right station at the line to be assembled.

Modern cars are built to customer specifications and this variety is part of the challenge in the automotive industry. The cars are manufactured at a rate of around one car per minute and at prices that most people can afford. The car itself is an imposing piece of engineering, but the industrial system that produces it is of a

complexity that is truly mind-boggling. The ability to mass produce individual cars is an example of mass customisation by *configuration design*, where the product structure is set up specifying what parts should be used, how they should be assembled and what variety is planned for. Configuration design in industry is normally executed with the help of some sort of design automation system, which in this context is called a *configurator*. The main purpose of the configurator is to automatically produce specifications of the product structure and technical specifications for all components whether manufactured in-house or bought from outside suppliers.

Motor car production was for a long time associated with standardisation. The efficient production systems that produce motor cars in large volumes and at modest costs initiated the paradigm of mass production in the nineteen-twenties. Standardisation allowing manufacture of very long series of identical cars is a central theme for this paradigm. During the nineteen-nineties the paradigm was gradually replaced by mass customisation, which implies that the benefits of long series production has been supplemented by a carefully controlled degree of variety so that the products can be adapted individually for each customer. It also means that production is not started until an order is placed—nothing is built to stock which reduces the costs of inventory and makes the company less exposed to risks of unpredictable market conditions.

Apart from satisfying individual customer requests, the adaptability of a configured product allows special product versions for different markets in the world as well as focusing segments of the home market, thereby giving deeper market penetration and higher volumes.

3.3.1 Modularised Products

By allowing the customer to specify products that to some extent are tailored to each customer's requirements, the product value as well as the possible market share are increased. However, if conventional work procedures are used it is likely that such customisation will mean increased costs and lead times, increased technical risks and a reduced efficiency in production, logistics and maintenance.

To be efficient the whole industrial process, including the design of the product, must be planned in a different way in order to create product families rather than singular products. For system products this leads to the use of the concept of product modularisation. These systems consist of modules and interfaces, where the product has been decomposed into units with well defined interfaces according to a deliberate strategy related to marketing, design or manufacture. Ideally the functional requirements should map to the specified modules as 1:1 to allow easy updating or maintenance. The interfaces must be carefully planned, flexible, well documented and have long life. New modules must be "backward compatible" so that updating of the system is possible with minimal parts replaced. When planning a modular product one needs to consider development, production, variety,

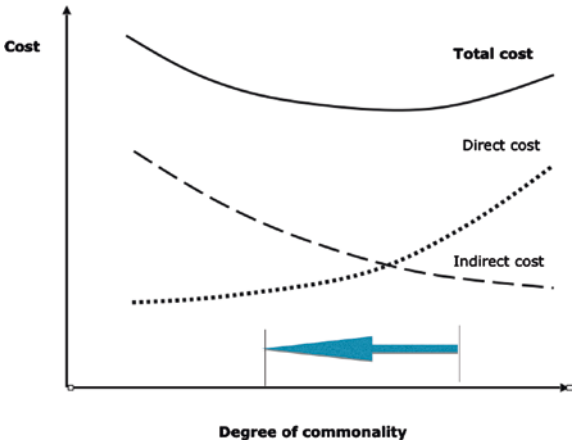


Fig. 3.8 Degree of modularisation expressed as commonality has an optimum where the total cost is lowest. A new product range should normally start at a high commonality but will over time tend to loose commonality due to market pressure, as indicated by the *arrow above*

logistics and service. For many of these aspects it is beneficial to arrange the product structure so that variant forming will take place late in the production process, i.e. during a large part of the assembly process the products in production will be identical.

In the automotive industry the concept of *Product architecture* is often referred to. This is a scheme for how the functions of the product should be allocated to its components (example: should the engine cooling water also supply coupé heating?) A group of related products that share common features, components, or subsystems; subject to a marketing plan, is called a *Product family* (example: PC frame that can be equipped with different CPU cards, graphics and storage facilities). These two terms describe how the product range is composed from a design and marketing perspective while the *Product platform* has more of a production and purchasing perspective ("exploit commonality") and defines the set of features, components or subsystems that remain constant from product to product, within a given product family. This might be the case for a family of car models including sedan, sports and estate models sharing e.g. drive lines, chassis and other subsystems. All three concepts have a strong bearing on the possibilities to deliver custom made products efficiently.

Modular design is a method to create product variety. But it is also a tool of more general benefit in that overhead, or indirect costs, are reduced. Indirect costs of typical engineering goods manufactured i modern highly automated factories, account for a surprisingly high portion of the total manufacturing cost, often more than half. Hence, reduction of indirect costs is high priority. On the other hand standardised modules will have a tendency to get higher specifications than what is necessary, resulting in higher component costs. The degree of modularisation chosen is thus something of an optimisation problem as illustrated in Fig. 3.8.

Efficiency in mass customisation by configuration design relies on three corner stones: Flexible production systems, flexible product structures and computer support to manage the process initiated by each new order. In this text we will focus on the two last aspects.

3.3.2 Product Structure for Variant Design

The potential for variety inherent in the product structure is one of the key issues for mass customisation. The objective is to meet customer demands of tailor made products by a clever set of options using a small number of components, while most parts of the product, the product platform, remains standard and made in long series, Products that to a large extent are assembled from modules are called *modularised*, while products that do not have a modular structure are called *integral*. For some system product modularisation comes naturally, while for other products this is not so and modularisation might result in over specification with heavier and costlier products as result. One obvious example is when many functions need to be carried out by the same component which is contradictory to one of the basic ideas of modularisation.

The product structure can be represented by a directed, rooted tree graph (see Sect. 5.2). The root is from praxis, although illogically, drawn at the top and indicates the complete product, the leaves the individual parts and the branches modules or subassemblies.

In the example in Fig. 3.9, two separate product structures are required—one for each tractor variant. For a family of tractors we would prefer to use a single, generic tree structure with a built in variability. This can be achieved by using a tree structure called Product Variant Master, PVM, which is a structural formalism specifically intended for configuration design, see (Hvam et al. 2008).

In the same way as for other product structures the links of the PVM tree denotes *Part of,* if followed upwards, and *Consists of* when followed downwards. The graph is constructed by considering each component one at a time and position it accordingly into the tree. To introduce variability into the product structure a link denoting *kind of* relations is introduced. The two *static* structures of Fig. 3.9 can then be merged into one flexible structure, as shown in Fig. 3.10.

According to the given specification the generic DFV is then *instantiated* so that parts and modules are chosen and the complete configuration defined. In many applications it is not possible to combine all options with each other. For the drive line for instance, the choice of engine must be matched by a suitable type of transmission. To define which restrictions that should be enforced one could use a relational table, e.g. like in Fig. 3.11, where all allowable combinations are listed. The computer system managing the product structures will then issue a warning to the user if a combination is chosen that is not listed in the relation table.

Often however the restriction on options are much more complex than what can be managed by a relational list. There might be relations between design variables that must be upheld, there may be conditional rules that must be tested and implemented and so on. For this purpose the DFV tree is complemented by a list of constraints, that define the allowable design space as regards configuration options. With this functionality implemented the computer system becomes a

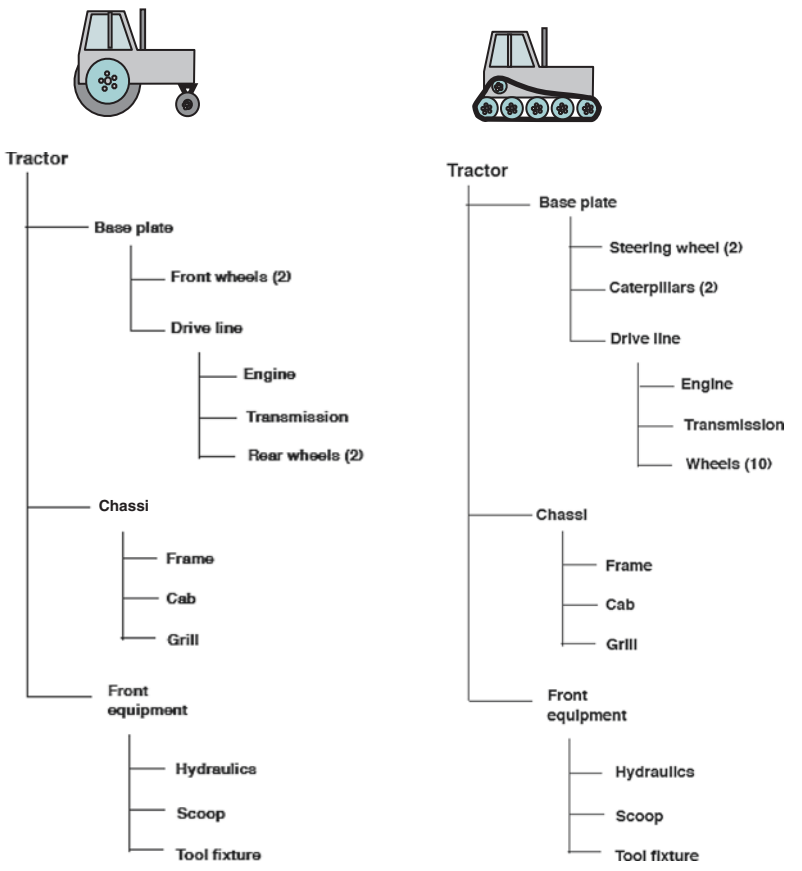


Fig. 3.9 Product structures as rooted trees for tractor with wheels (*left*) and caterpillar traction (*right*). Going downwards in the trees defines "consists of", going upwards defines "part of". Adapted from Elgh (2009)

product configurator. Depending on the complexity and amount of constraints the constraint handling capacity could reach from a simple decision tree with conditional restrictions to a powerful constraint processing system of a generic character where constraints can freely be inserted or removed.

An alternative graphic representation to PVM is to use some standard notation for object oriented modelling like Unified Modelling Language, UML, see (Haug 2008). This is generic and provides a rich expressiveness, but might feel less natural to most engineering designers. In this text we will therefore use the PVM approach.

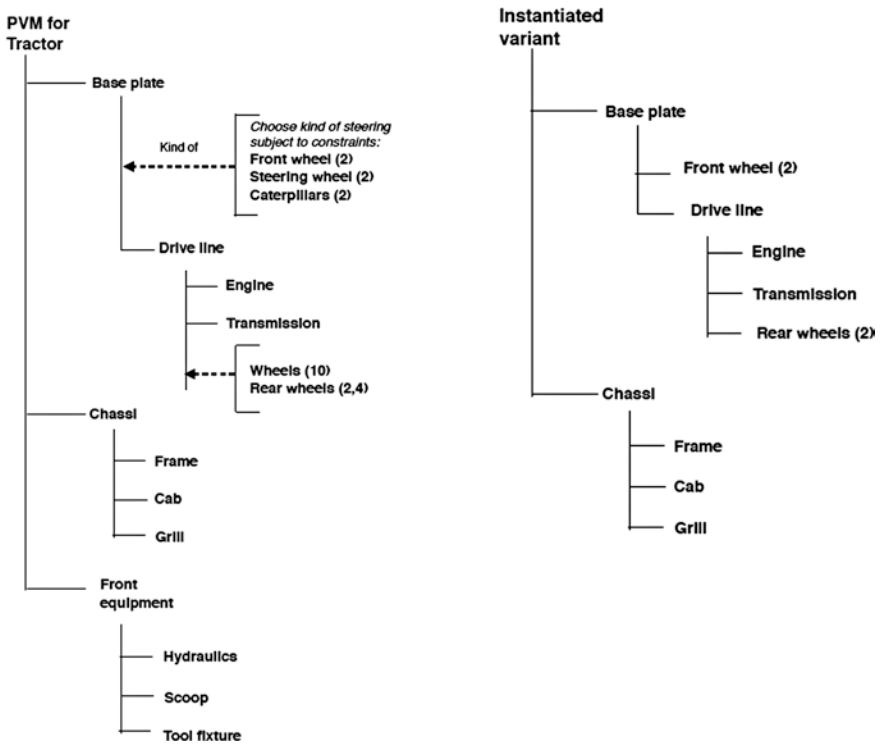


Fig. 3.10 Flexible product structure in the form of a PVM for generic tractor (*left*) and instantiated as a four wheel tractor without front equipment (*right*). In the PVM alternative choices of components ("kind of" relations) have been denoted by *dashed lines*. These choices to be made subject to constraints, e.g. front wheels cannot be selected if caterpillars were chosen. From Elgh (2009)

Fig. 3.11 Relational list of allowed combinations of components in motor car drive line

Car	Engine	Gearbox	Country
V70	HP120	GB_Xv1	SWE
V70	HP125	GB_Xv2	SWE
C70	HP120	GB_BBc	UK
V40	HP160	GB_Xv1	SWE
C70	HP160	GB_Xv1	UK
V40	HP120	GB_Sp1	SWE
…	…	…	…

3.4 Generative and Hybrid Design

Building a system product by configuration of a set of predefined components is often referred to as *catalogue design,* a common approach in design of electronic equipment, hydraulic systems and other products built from bought components. There is however also the possibility to combine configuration design with parametric design, e.g. *configuration of parametric components*. The parametric components that are intended to be part of a configurable system product, are sometimes referred to as *configurable components* (Classon 2006). These *hybrid design* systems could be applied to products constructed of components that are all parametric or a mixture of parametric and static components. The term "component" does not necessarily refer to a separate, physical object, it could also apply to functional features of a part or even be an abstract part like a cavity designed for a specific purpose. For the part dimensions to be variable it is required that constraints are defined so that changes of the driving variables propagate to all dependent variables in order to maintain the intended geometry characteristics. Figure 3.12 shows an industrial example of a product family for cutting tool heads that are configured from parametric parts.

Parametric and configuration design represent different design methods. However, if the steps within a component range with *kind-of* relations to the PVM structure are made very small, the configurator will deliver design solutions similar to those of a parametric design system. In this way the two methods overlap so that systems built according to the principles of configuration can also handle parametric problems and vice versa.

In parametric design as well as in configuration design the solution space in terms of allowable variable ranges, product structure(s) and constraints is clearly defined. For a given requirement specification this solution space is searched

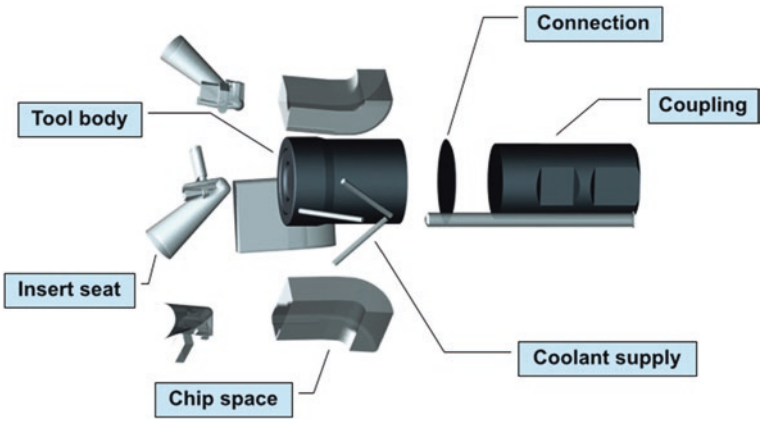


Fig. 3.12 Parametric parts used to configure a cutting tool head. See also application case No 2 in Appendix. Courtesy of Sandvik Coromant AB

for one or several *legal* solutions that satisfy all constraints. If such a solution is found, these design variables are inserted into the product model and a proposed solution is *instantiated*. This is straightforward and a common approach to design automation, but there are also cases when this is not possible or practical.

Some problems require that a proposed solution is generated step by step during program execution where subsequent design steps depend on previous ones using some form of recursive algorithms. Not until the whole design is completed is it possible to know whether all constraints are satisfied and a viable solution has been found. We call this approach to design automation *generative design*, which implies that the design proposal is generated under program control rather than being instantiated from a search tree representing the legal solutions.

Take as an example application No 8 in Appendix, the lay-out of resistance wire in a car seat heater. The lay-out is subject to constraints and it has an objective function (cost) that should be minimised. Fundamentally this is an optimisation problem but at present stringent optimisation was considered not feasible. Instead lay-outs were created using experience based rules for a sequence of varying inputs. The proposed lay-out was then tested for adherence to constraints and

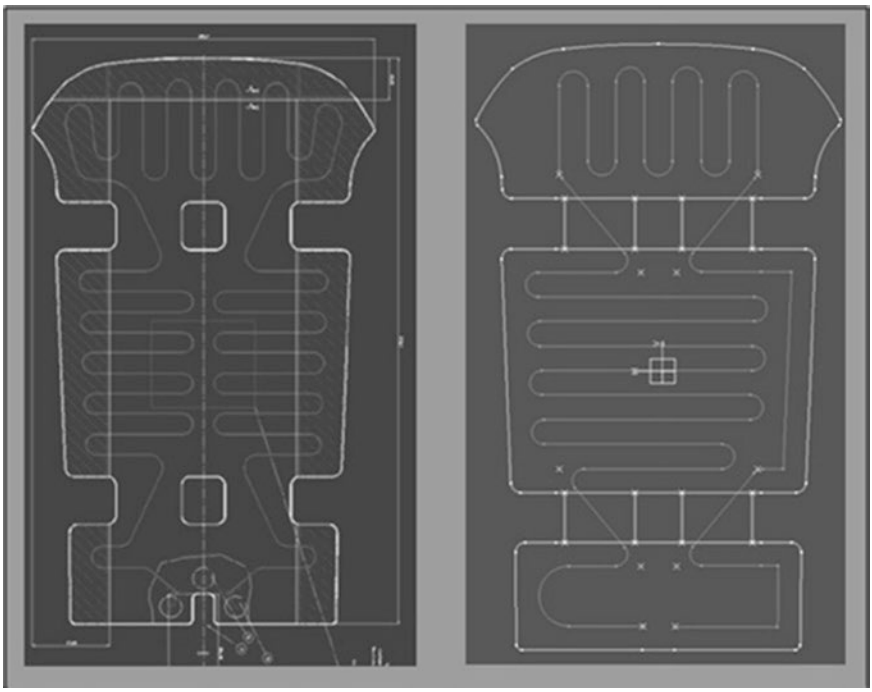


Fig. 3.13 Seat heater wire lay-out. Manual design (*left*) automatic design (*right*). Both are considered satisfactory from a user point of view. From Elgh (2009). Courtesy of Kongsberg Automotive AB

accepted or failed. The solution that satisfied all constraints and had the lowest cost while satisfying thermal requirement specification was selected as the best design. This best design was used for cost estimates and quotations. For the orders that the company wins, it is usually possible to improve the design somewhat further manually (Fig. 3.13).

References

Amen, R., Sunnersjö, S.: The Solids Handbook (in Swedish). The Association of Swedish Manufacturing Industry, Stockholm (1996). ISSN 1103-7067

Cederfeldt, M.: Planning design automation. Ph.D. thesis, Chalmers University of Technology, Göteborg, Sweden (2007)

Classon, A.: A configurable component framework supporting platform-based product development. Ph.D. thesis, Chalmers University (2006). ISBN 91-7291-791-1

Elgh, F.: Lecture notes from JTH masters program in product development (2009)

Haug, A.: Representation of industrial knowledge as a basis for developing and maintaining product configurators. Ph.D. thesis, Technical University of Denmark (2008). ISBN 9788791035678

Hvam, L., Mortensen, N.H., Riis, J.: Product Customisation. Springer, Heidelberg (2008)

Chapter 4
Clarifying, Idealising and Modelling of Engineering Knowledge

As human beings we think and reason without paying much attention to our own mental processes. Human intelligence appears to be a collection of mental abilities that are partly complementary but also overlapping, creating a degree of redundancy. The challenge when setting up a design automation system is to transform knowledge in a form that has been satisfactory for the workings of the flexible and resourceful human mind into facts, rules and methods that can be programmed for execution by a rigid and stringent computer processor. This includes processing *explicit* and *implicit* knowledge or, when possible, making implicit knowledge explicit, to device methods to *bridge knowledge gaps* when such occur or, conversely, to mediate when design rules are *contradictory* or *overlapping.* This chapter will discuss the most important aspects of this process.

4.1 What Is Knowledge?

Knowledge processing and knowledge representation are essential elements in design automation, which is sometimes also called *Knowledge Based Engineering*, *KBE*. While this term might be a somewhat restrictive characterisation it rightly emphasises how central the concept of knowledge is to design automation. Before proceeding we thus need to discuss what we really mean by knowledge in the present context.

Knowledge theory is a central field of philosophy called *epistemology*. This theory discusses questions about the nature of knowledge and what the foundations of knowledge are. Philosophers have for 400 years been divided between the two main approaches of *knowledge from experience* (Empirism) and *knowledge from reasoning* (Rationalism), where the former believes that all knowledge is obtained from the surrounding world via the senses, while the latter means that all knowledge

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_4

come from abstract reflections upon ideas. To build knowledge based system we need some sort of mental framework for the concept of knowledge, but the study of knowledge in the philosophical sense is far beyond the scope of this work.

To arrive at a useful definition of such a framework we need to narrow down the scope from knowledge in general to knowledge about automated engineering design in particular. A tentative, but practical, taxonomy for this class of knowledge will be presented in Sect. 4.4. Also the discussions below refer to this narrow scope.

The insights that an individual, a department or a company have about a given product technology can be seen as a hierarchy in three steps:

- **Data**. Static, unorganised and unprocessed facts
- **Information**. Data systematically aggregated for a purpose
- **Knowledge**. Understanding of information based on perceived importance and relevance for a problem domain. Includes insights, ability to apply methods and information, perception, skill, praxis, training, experience

In this text the term *design knowledge* is used for knowledge according to the definition above, but delimited to what is required for a specific design task.

There is a slight problem with the word "understanding" in the definition of knowledge. Since a computer can have no understanding in any sense, how is it possible to implement knowledge in a computer at all? If we see this from a strictly semantic viewpoint the answer is: No, it isn't possible. On the other hand, if we have knowledge of a procedure that we understand will solve the task and if we can program the computer to carry out this procedure, it seems perfectly reasonable to state that we have implemented knowledge in a form of understanding of how to carry out a certain task.

As was mentioned previously the term *Knowledge Based Engineering* is sometimes used as a synonym to *Design Automation* in general and *Inference based* systems in particular. Taking into account that engineering problems often involves also procedural, mathematical processing, the less restrictive term Design Automation has however generally been used in this text.

4.2 Thinking Inside the Box

In product development there is a time for creativity and innovation and there is a time for constraints and standardisation. The engineering design profession attracts people with an inclination for creative work and pursuit of excellence, so standardisation and other restrictions do not come naturally. In product development projects there are however milestones when decisions need to be taken and design solutions frozen. Once such a milestone is passed, new ideas that might replace the already selected solutions are, generally speaking, not welcome. This is so because subsequent design work will be based on earlier solutions and the project will never make any progress if early decisions are frequently altered.

Design automation is based on standardised procedures. There are things to be said for such standardisation in general. If specific design tasks are always carried out in the same, predefined way it will generally result in a better and more consistent quality. The work will not be critically dependent on which person that has been assigned the task and more effort can be invested in establishing the procedure. Also, what the Japanese call *Kaizen*, i.e. small, frequent steps of improvement, is only possible when a standardised procedure is available to start from.

The same is to some extent true also for new versions of products already on the market. Of course modifications and improvements need to be introduced when a revision of an existing product is to be made. It is however important to restrict the alterations to certain modules or components and leaving the remainder intact, otherwise the cost of the product will escalate out of proportion. Designers will always feel tempted to introduce improvements in existing products, but such changes must be introduced in a well-planned and disciplined way.

There is thus a delicate balance to strike between innovative and more evolutionary solutions in product development. Innovative solutions mean that new knowledge often must be sought, while thinking of products within the confines of well-known and proven technology is based on established and available knowledge. Design automation is about the latter kind of tasks, but with the intention of freeing time and resources for humans to focus on the former.

4.3 Idealisation of Product Knowledge

The process of transforming human product knowledge to executable code is usually extensive and gaps or weaknesses in existing knowledge are often revealed. The approach based on idealisation and modelling is familiar to anyone with experience from setting up simulation programs for physical phenomena in technical systems. This approach is well applicable also for creating a simulator for design office work. The approach is top-down in the sense that it starts out with clarifying and condensing domain knowledge (idealisation) and proceeds with determining a suitable method for representation (modelling). It is often also clarifying to determine the characteristic problem structure (see Chap. 5), which indicates which method(s) for knowledge processing that could be suitable.

In the traditional design office the product designers usually have many years of experience of their products and how they are manufactured. The design knowledge is a mixture of experience, stringent rules, company policies, rules of thumb and so on. For the person in charge of building a DA-system it will soon be clear that it is not simply a matter of interviewing designers to extract design rules. Because of the experience and versatility of the engineers, a stringent and exhaustive set of design rules have usually never been needed before and consequently does not exist. Steps to handle this situation are discussed below and in Fig. 4.1.

There usually exists domain knowledge for the technical aspects of the product that is accessible but unstructured. We can call this “raw” knowledge, which

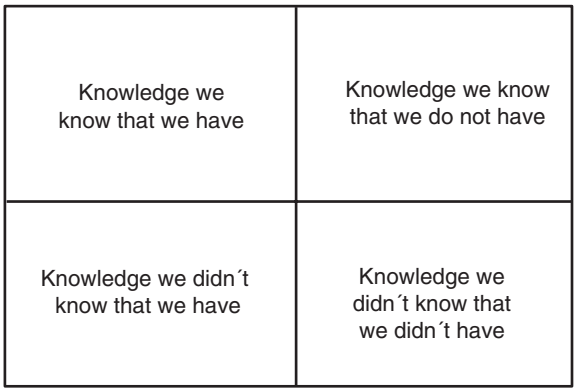


Fig. 4.1 Classes of knowledge encountered during knowledge acquisition. Adapted from Luft and Ingham (1955)

consists of various sources of information related to the product and its manufacture. The raw knowledge is likely to have gaps where the required knowledge is lacking as well as contradictory knowledge elements. Despite these shortcomings of available knowledge, the necessary decisions have been taken, usually out of experience or intuition.

Building a design automation system provides an excellent opportunity for the designers to clarify knowledge gaps and contradictions and agree on what should be company praxis. The benefit of this work thus does not only support building a Design Automation system, but often equally important, it improves the quality of what we might call *corporate knowledge*, which may well be the company's most valuable asset. By corporate knowledge we mean such knowledge that is explicitly defined and documented in a form that makes it available to all individuals in the organisation that benefit from it to fulfil their given tasks.

Knowledge acquisition is primarily directed towards the top left quadrant of Fig. 4.1, i.e. towards such knowledge that the organisation is aware of possessing and using. In this process knowledge is often also discovered in the lower, left quadrant, which represents knowledge that is familiar to singular individuals but should be useful also for others in the organisation, i.e. individual knowledge should be made part of the corporate knowledge. Similarly, knowledge gaps may be discovered relating to the upper, right quadrant for knowledge which the company decides it will not need. This could e.g. mean that certain manufacturing methods or certain materials intentionally have been excluded. Finally, and this is what makes design managers nervous, the lower right quadrant relating to knowledge one did not know one did not have or knowledge one thought one had. Of course it is of the utmost importance to identify such knowledge gaps as early as possible, not during prototype testing, or even worse, not after delivery to the customer. The costs to rectify faults escalates quickly with time.

The raw knowledge needs to be idealised in order to form a basis for a computerised knowledge base, see Fig. 4.2. Hence, from the large and only partly relevant source of "raw knowledge" we need to:

- Select the relevant parts
- Formulate clear design rules based on this knowledge

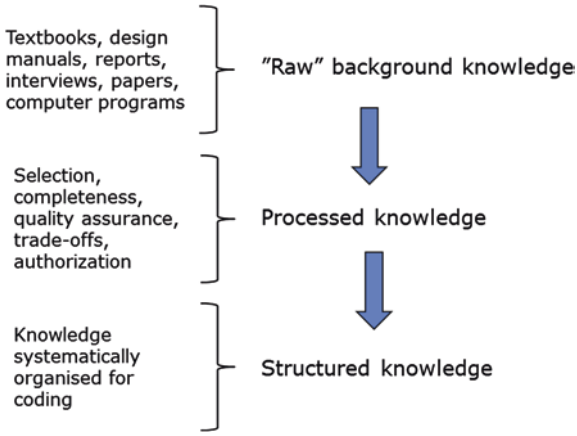


Fig. 4.2 Idealisation of raw knowledge to knowledge ready for coding into DA system

- Investigate whether the set of design rules is complete and consistent
- Verify the correctness of these rules
- Define compromises between conflicting rules
- Develop new knowledge where knowledge gaps appear
- Document design rules
- Ascertain that the knowledge base represents the design intent of the design team
- Verify by a formal authorisation process that the rule set does represent corporate knowledge

The outcome of this process we can call "processed knowledge". This knowledge will subsequently be coded to form the knowledge base of the DA system. For this purpose the processed knowledge is cast into a format that is in accordance with the syntax of the DA system used. The structured knowledge should still be human readable, but structured so that computer coding comes naturally as the next step. It also helps documentation and future maintenance if the knowledge base is structured in a way that appears natural for the designers.

4.4 Characterisation of Engineering Knowledge

The classification presented below should be seen not so much as a stringent, quantitative model of reality, but more as a helpful mental framework when planning design automation. In order to identify suitable methods for knowledge representation it clarifies the problem when the knowledge is categorised into meaningful groups. Each group or category in this classification maps naturally to frequently used knowledge representation methods. Six classes of product knowledge for DA defined for this purpose were given in (Sunnersjö 2009, 2010). The different categories represent knowledge with varying richness and stringency

of background knowledge. It is to be expected that as design rules and methods evolve and mature over time there will a natural progression of knowledge towards the more stringent categories.

The empirical basis for this study of knowledge categories is investigations at ten manufacturing companies in Denmark and Sweden. The applications represent high industrial variety, where some applications focus on advanced high-tech products that need to be highly optimised; others focus on tailoring each product to individual customers or to prepare layouts for quotation calculations. The six classes are:

1. **Tacit**[1] **knowledge.** Unexplained and unarticulated knowledge embedded in the human mind through common sense and experience. Includes intuition, convictions, skills, craftsmanship. It is generally recognised that tacit knowledge cannot easily be communicated since it is not explicit. The concept of tacit knowledge was introduced by Polanyi (1966).
2. **Knowledge based on comparison.** Guide lines based on experiences and insights from comparable products and processes. The design rules are often not articulated explicitly but are embedded in the design of previous products. If previous experiences and lessons learned are stored, understood and taken into account the idea of basing new products on existing, proven solutions is attractive. However, previous experiences might not be applicable or misunderstood which can lead to unexpected malfunctioning of the new product.
3. **Experimental knowledge.** Facts and relations that are reliably established by experiments, e.g. measured physical properties, performance, efficiency and so on. Within the bounds of the empirical investigations interpolation from measured data to required design data is often acceptable, while extrapolation outside the bounds of experience should be avoided.
4. **Geometrically related knowledge.** In mechanical engineering spatial relationships and geometrical reasoning is often of great importance. Function is often achieved through the geometrical form of products, i.e. function is embedded in form. Examples where geometrical reasoning is a key factor are styling, packing problems, complex assemblies, load carrying structures and designs where action of flowing media is involved.
5. **Knowledge represented in mathematical form.** When governing design variables can be derived from fundamental physical principles the problem can be represented in a mathematical form. For simplified problems explicit analytical expressions may be available, but for most realistic design problems approximate, numerical computations are required. Design methods in the form of mathematical expressions or numerical algorithms provide physical insight and rigour to the problem and are therefore preferable when available and realistic in terms of effort, computer execution time and so on.

[1]From Latin Tacitus, meaning “silent”.

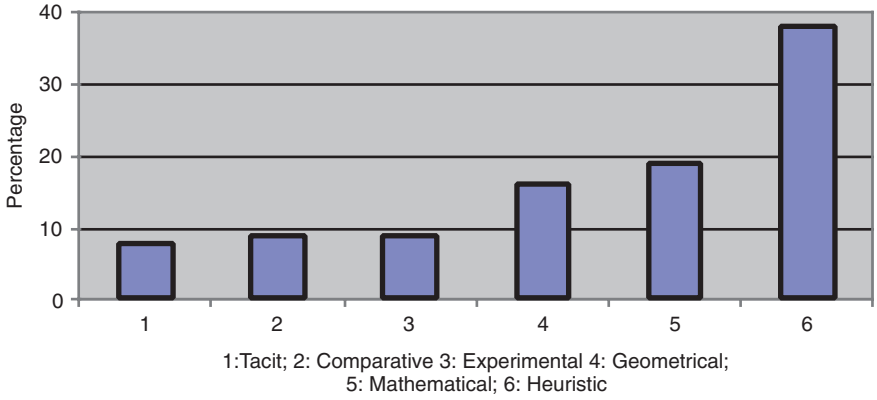


Fig. 4.3 Accumulated distribution of knowledge into six categories for the ten industrial application cases studied

6. **Heuristic knowledge.** Simple and useful guide lines that are based on experience, reasoning and fragments of theory. Knowledge expressed as facts and rules. Not necessarily based on stringent theory and consequently give poor insight into phenomena and governing parameters. Application outside range of experience may result in unpleasant surprises. When groups of authorities agree on heuristic rules, design praxis is defined. Praxis is a common and important form of documented knowledge; more specifically, many commercial and legislative standards (e.g. ship classification rules, pressure vessels codes, standards for lifts and trains) belong to this category of knowledge.

The accumulated distribution of the ten companies into the six knowledge categories is plotted in Fig. 4.3. Heuristic knowledge dominates clearly, with mathematical and geometrical knowledge coming second and third.

4.5 Modeling of Knowledge for Computer Processing

4.5.1 The Computer and the Human Brain

In Sect. 1.3 the approach of seeing the design automation system as a "simulator of the design office" was introduced. Similarity behind cognitive processes in humans and computers appear however to be rather superficial, which is not surprising considering how different the "hardwares" are. The explicit, step-by-step reasoning performed by the computer is not typical for the human way of thinking. Instead reasoning goes on between individuals and through inner monologs, while decisions often are taken as instant response based on experience, intuition or pet ideas. Humans, including human engineers, do not behave as logically as

we would like to think we do. So, it is not self-evident that the computer system should attempt to emulate the human cognition process—there might be better ways to solve a problem. What is clear however, is that the results of the computerised process should coincide with (not necessarily be identical to) that of the human expert. Often it is within reach to develop a computer system that produces consistently better solutions than the manually produced counterparts.

In order to appreciate the magnitude of the challenge to represent human thinking within the confines of such a blunt instrument as the digital computer we shall review a few facts about the human brain and how it compares with the computer (Balkenius 2007). For an exhaustive treatment of neural science, see a standard textbook like (Kandel et al. 2000).

Our understanding of the human brain has improved dramatically during the last decades, much due to refined methods of computer tomography. Still, a real understanding of how the brain actually works on a detailed level does not exist.

Mental processes are performed in the cerebral *cortex*, which is a thin layer of *neurons* covering the surface of the brain. The part of cortex where abstract thinking is carried out is positioned in the frontal lobes. The interior of the brain mainly consists of connections, *axons* and *dendrites*, between the neurons plus support systems to provide the neurons with oxygen and nutrition and to provide structural stability and a suitable working temperature.

The brain has approximately 90 billion neurons in cortex, all directly or indirectly connected. This represents an enormous capacity and, unlike the computer, the human mind never runs out of memory. The brain has grown out of the primary task of managing our bodies. Abstract thinking, e.g. the ability to anticipate the future from lessons learned in the past, is from an evolutionary perspective a late and minor function. The brain capacity dedicated to eye-sight and image processing for instance, far exceeds the capacity dedicated to abstract thinking. However, the ability to reason about abstract things has late in the evolution of man, proved to be of great value and has strengthened our competitiveness relative to other species in such a way, that we have become the undisputed rulers of our planet.

The functions of the brain rely on electro-chemical signals. These signals are very slow compared to the electrical signals of the digital computer. Nevertheless the human brain can respond extremely fast when required, e.g. in memory retrieval or response when something threatens us. The explanation is the extensive interaction between the interconnected neurons. The brain neurons work using massive parallel processing on all levels to an extent that no super-computer can challenge. Realising this characteristic leads to the conclusion that digital computers outperform humans when it comes to serial processing like heavy number crunching, but humans are much more efficient in finding associations, comprehend phenomena, find new solutions and so forth. Of course these are differences that we should keep in mind when selecting tasks for design automation.

One striking feature of the human brain is its ability to work with and draw meaningful conclusions from *incomplete knowledge*. We have the very important ability to bridge knowledge gaps in a sensible way. Consider Fig. 4.4. Despite the fact that no sides of the triangle are drawn, we interpret the geometry as if a white

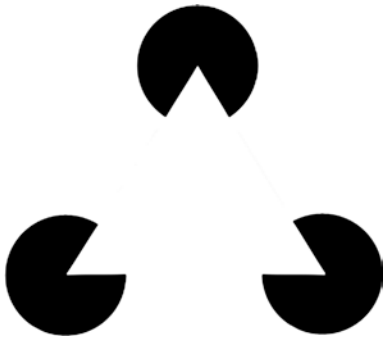


Fig. 4.4 Three *black circles* with *cut-outs*. The eye sees a *white triangle* laid over the *circles*, but the *triangle* does not really exist. From Kanizsa (1955)

triangle partly intersects the three black circles. We do this because in our experience this is usually the case when we see a pattern of this kind. For the human being quick conclusions from incomplete information is a necessity otherwise we would be too slow and too hesitant in our response to events that affect or threaten us.

Is this ability good or bad in an engineering perspective? It could be both: On the one hand it is a very fast and efficient method if missing information can be sensibly filled in. On the other hand, jumping to preconceived conclusions could lead seriously wrong. The fact that authorities agree on conclusions from incomplete knowledge should not be mistaken for stringent knowledge—the reliability of the knowledge used is still the crucial factor. One thing is certain however, the computer demands 100 % complete information and cannot draw any conclusions at all if this is not the case. Every small detail, however obvious it might appear, must be specified in the computer code. For the computer nothing is self-evident!

The human mind is also able to deal with the opposite situation, namely *redundant, contradictory and overlapping information*, or, in our case, design rules. When confronted with design rules that are contradictory we can usually reach a sensible compromise or give priority to the most relevant or convincing design rule. Such applications of experience and common sense is not within the powers of the computer. There are strategies available though that goes some way towards this goal, e.g. defining weights for the rules to determine which should be given priority, or giving preference to rules dated late or rules that statistically have often been referred to. When such strategies do not seem relevant, one must resort to letting the computer ask the human operator for a decision.

The conclusions from the brief summary above is that computer based design automation systems are best suited for problems that are repetitive and laborious by nature and require attention to detail as well as endurance and access to a large knowledge repository. Examples of this could be long series of numerical computations or retrieval of information from large data bases. The human mind on the other hand is superior for problems requiring comprehension, experience, common sense, good judgement and so forth. It is wise to keep these fundamental differences in mind so that the computer is applied to the tasks for which it is best suited and not compete with the human mind where it is at its best.

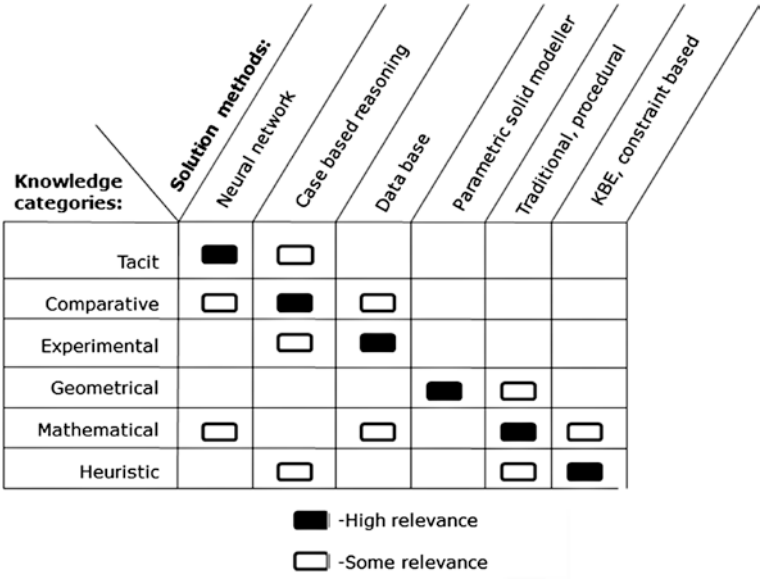


Fig. 4.5 The matrix indicates suitable implementation methods for different categories of engineering knowledge when planning systems for automatic design

4.5.2 Representation of Knowledge

DA-systems are often based on technologies developed in the field of *Artificial Intelligence, AI*, which are technologies developed for the purpose of representing, or modelling, human reasoning and knowledge. The more commonly used methods in design automation were briefly introduced in Sect. 1.4, see in particular Fig. 1.4. We will now proceed with a discussion of how the six categories of engineering knowledge introduced in Sect. 4.4 should be matched to these methods of representation.

The different knowledge categories previously described have a corresponding software method that is well suited to that particular type of knowledge as illustrated in Fig. 4.5. This mapping between knowledge category and modelling tools will be the main topic for the following Chaps. 6 and 7.

References

Balkenius, C.: Intelligent computer systems for automated engineering design. Lecture notes for Ph.D. course. JTH, Jönköping, Sweden (2007)

Kandel, E., Schwartz, J., Jessel, T.: Principles of Neural Science. McGraw-Hill, New York (2000)

Kanizsa, G.: Margini quasi-percettivi in campi con stimolazione omogenea. Rivista di Psicologia **49**(1), 7–30 (1955)

Luft, J., Ingham, H.: The Johari Window. University of California, Los Angeles (1955)
Polanyi, M.: The Tacit Dimension. Doubleday & Co., New York (1966)
Sunnersjö, S.: A taxonomy of engineering knowledge for design automation. In: Proceedings of TMCE 2010, Delft University of Technology, The Netherlands (2010)
Sunnersjö, S.: An empirical study of different aspects of knowledge used in engineering design. In: Proceedings of NordPLM, Göteborg (2009)

Chapter 5
Problem Structure and Knowledge Processing

5.1 Well-Structured and Ill-Structured Design Tasks

To solve a computable design problem a sequence of operations (arithmetics, data storage, evaluation of conditional statements and so forth) will be performed. There are cases where this is a straightforward task, the sequence of operations are well known or can be clearly established. Often this is the case for processing associated with analysis. Such problems are said to be well-structured, implying that the program code has a predefined path of execution, which it is up to the programmer to clarify and specify.

Let us look at an example of a well-structured problem. Assume the task is to write a program to calculate the cross sectional inertia, function I(Fb, Ft, Lh, Lt), of the beam cross section of Fig. 5.1, with respect to the dotted baseline. The contributions from the two flanges and the web are calculated according to standard formulas and then these three contributions are translated to the baseline using Steiner´s theorem. The resulting formula is written top right in Fig. 5.1. All parameters are available in the sketch, but we can see that the vertical position of the centre of gravity, function TP, is required and needs to be calculated. This is easy enough and also done using standard formulas with the result given at the bottom right of Fig. 5.1. However, since TP is needed to calculate I the order of the two formulas must be reversed. For this calculation the causality between TP and I will always be fixed and the same. This is a type of calculations that are best programmed by traditional, procedural programming methods and where the causalities between different steps of the processing are naturally imbedded in the structure of the code.

Design tasks however, often require a synthesis process and typically these problems are more open and flexible than an analysis process. Design processes are by nature creative, have no fixed solution path and often many possible, valid results. Many synthesis processes are inherently *ill-structured,* where program control

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_5

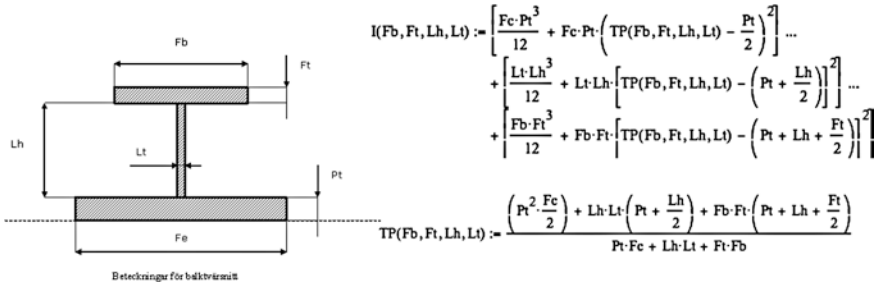
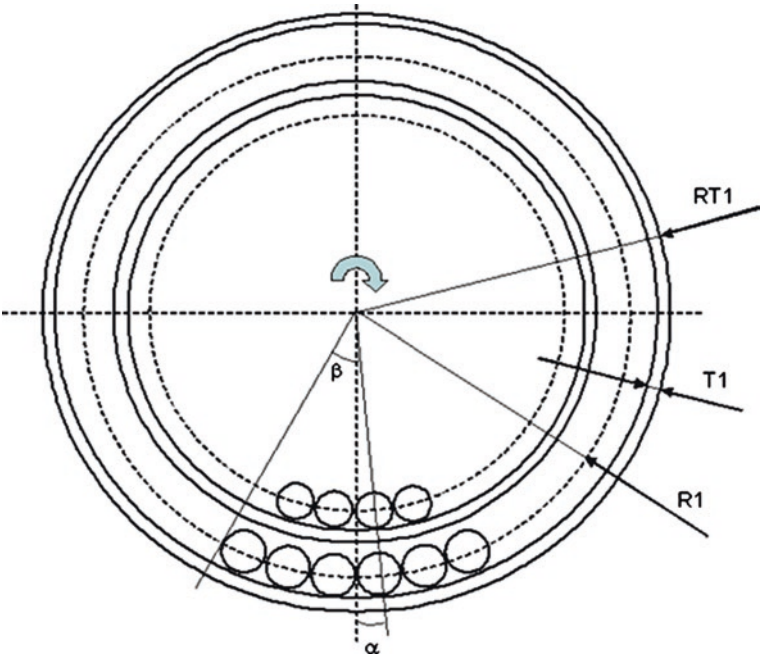


Fig. 5.1 Calculation of cross sectional inertia of I beam

needs to be determined during execution rather than following a predetermined path towards a solution. This type of problems are usually solved by some kind of *search based* approach often using methods from the field of Artificial Intelligence.

Typical applications of ill-structured problems, which require search based solution methods, are scheduling problems (e.g. match classrooms, teachers and classes for a semester at university), planning tasks (e.g. plan material flow and manufacturing operations in a factory plant) and packaging (e.g. fill parcels of different sizes into a container). Here and in Chap. 6 we will use a more design oriented problem to exemplify problem structure and solution principles, namely the design of a machine element called "automatic balancer". This is a device used in high speed rotating machinery to counteract the effect of occurring unbalance. Typical applications are handheld grinding machines, laundry spin driers or blood centrifuges. The principle lay-out of the element is shown in Fig. 5.2. Due to fundamental dynamic phenomena the balls will automatically move to positions where they counteract the external system unbalance when the rotational speed exceeds the critical speed of the system. The effect is to minimise rotor vibrations.

The balancer represents a typical "open ended" design task. It is designed and manufactured by a sub-contractor on order from many different companies, who fit it as a component in their own products. One such customer might want to know what balancing capacity a given design will have, another customer may want to know what diameter and how many tracks are required to reach a specified capacity and a third customer wants to minimise the ball diameters to keep the thickness of the device at a minimum. A large number of relevant sets of input parameters and relations to corresponding output variables can be set up, and the engineering knowledge required to answer these questions are well known. Ideally we would want to be able to use the same computer code for all variations of the design task. This means that a rather small number of relations and constraints that applies to parameters and variables should be executed in different sequences and with different causalities (directions) depending on what questions that are asked. With this requirement of *dynamic input parameter sets* the design problem becomes highly ill-structured. We will see how the four main solution principles for knowledge processing will perform in this respect. In order to clarify the problem structure in a stringent way, we will use methods from graph theory, which will be introduced in the next section.



Notation

Alfa	Ball angle	NN	Number of balls in track N
BMN	Mass of ball in track N	RT1	Outer radius of disc
Dens	Density of balls	RT1N	Outer radius of track N
FiN	Diameter of balls in track N	RN	Radius of balls in track N
MaN	Mass of balls in set N	TN	Wall thickness of track N
N	Number of tracks	UBM	Unbalance moment

Fig. 5.2 Dynamic balancer with main design variables. Product properties are total mass of balls and the resulting unbalance moment

5.2 Fundamentals of Graph Theory and Dependency Structure Matrices

To understand if, and how, a design problem can be solved it is essential to clarify and understand the structure of the problem, i.e. how the sub-tasks or processing steps depend on each other and how the information flows during processing. Depending on the problem and the intended implementation methods this could apply not only to the data flow during execution but also to the solution search space or the product structure.

There is a branch of discrete mathematics called “Graph theory” which is dedicated to the study and analysis of generic structures. This theory is due to the eighteenth century German mathematician L Euler. He was intrigued by the

problem of devising a route over the seven bridges of his hometown Königsberg so that the same bridge was never passed more than once. These Sunday walks speculations resulted in a formalism for structures which is applicable to many disciplines and widely used also today, see e.g. reference Haggarty (2002).

Graph theory is closely linked to the theory of *Dependency Structure Matrices, DSM,* which provides an alternative way of expressing the same information in a more compact manner and in a format allowing computer processing of the information, see references Warfield (1973), Steward (1981) and Eppinger (2012). Both methods serve the purpose of allowing complex problems to be represented in a way that allows conclusions about problem properties (subtasks, dependencies, causality, information flow) and applicable solution methods to be drawn.

5.2.1 Definitions in Graph Theory

In the following those elements of graph theory that are particularly relevant to design automation applications will be briefly summarised.

Definition A **graph**, G = (N, L), consists of nodes (vertices) and links (edges), see Fig. 5.3. The graph defines a finite domain, i.e. there is a finite number of discrete nodes and a finite number of discrete links.

A graph is essentially an illustration of a list of nodes and their connections as shown in Fig. 5.4. The connectivity of the nodes are defined by the list of pairs forming start and end nodes for each link. The arrow between Fi1 and R1 in Fig. 5.4 represents the link between these two nodes and is the first entry in the list of links. The information in the list and in the graph is the same, but the structure of the graph is more comprehensible than that of the list.

Definition Graphs could be **undirected** or **directed** depending on whether the links can be followed in both directions A directed graph has links that imply a

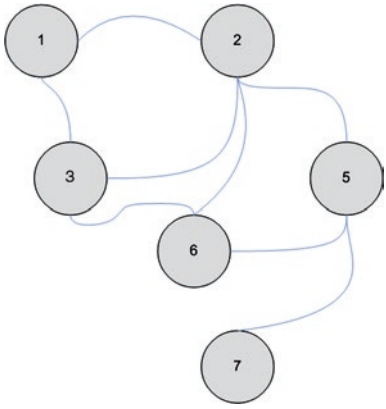


Fig. 5.3 A graph consists of nodes and links

Nodes (vertices):	Connectivity (links, edges, arcs):
T1	T1;R1
Fi1	Fi1;R1
N1	N1;UBM
R1	R1;UBM
Alfa	R1;Alfa
BM1	Alfa;UBM
UBM	Fi1;Alfa
	Alfa;BM1
	BM1;Fi1

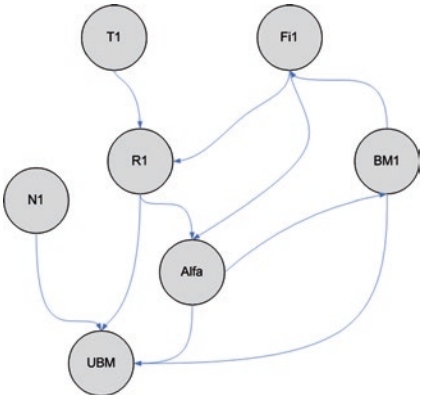


Fig. 5.4 A "graph" is a representation of a list of ordered pairs. The example shows a directed graph

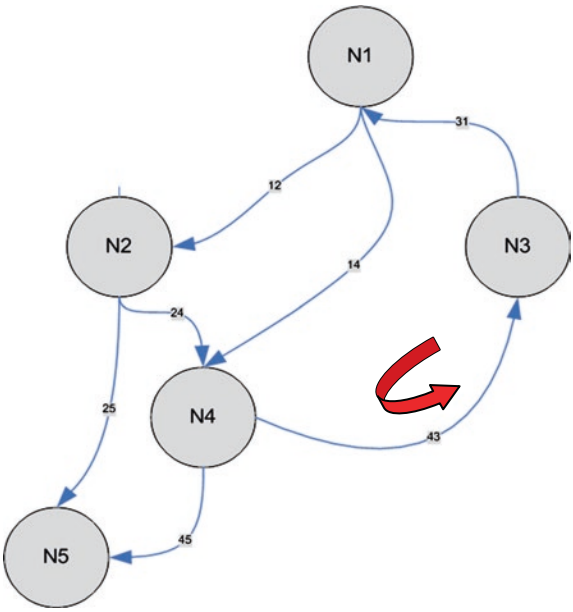


Fig. 5.5 Directed graph with one cycle

causality or a sequence in time that must be enforced. Figure 5.3 is an undirected graph and Fig. 5.4 is a directed graph.

Definition A **path** is a sequence of links connecting a pair of nodes, e.g. L14;L45 is a path from N1 to N5 in Fig. 5.5.

Definitions A path that returns to a previously visited node forms a **cycle,** see the path through nodes 1-4-3-1 in Fig. 5.5.

5.2.2 Tree Structures

For DA applications a special type of graphs called “trees” are of particular importance. Definitions, theorems and properties of tree structures are summarised below.

Definition A graph is **fully connected** if all nodes can be reached through a sequence of links (=paths exist between all pairs of nodes).

Definition A **tree** is a graph which is fully connected and has no cycles.

Definition A **rooted tree** is a tree graph starting in a single node. Nodes with only incident links (no links leaving the node) are called **leaves**. The tree graph is usually drawn upside down (root at top).

Theorem *For a **directed rooted** tree it is always possible to sort all nodes in an ascending order so that* i < j*, implying that for a data flow graph the determination of node i should precede determination of node j.*

Definition A graph sorted with all nodes in an ascending order is said to be **topologically sorted.** The corresponding DSM, see Sect. 5.2.3, has all nonzero elements **below the diagonal**. This implies that no variable value is requested before it has been determined

Definition The **directed tree** structure is used to represent **hierarchical** structures. Examples are **product structures** (rooted), **search trees** (rooted) or **data flow charts**

Definition A data flow problem represented by a directed rooted tree implies that the input parameters form the starting point or root. If one or several of these starting parameters are replaced by design variables, the problem is said to have a **dynamic input data set**.

5.2.3 The Dependency Structure Matrix (DSM)

For large problems with many nodes and links the graphic representation becomes complex and the overall view and clarity is lost. Further, the graph itself is not suitable for computer processing. A matrix notation in the form of an “adjacency” (binary) matrix is then often preferred. The adjacency matrix is a type of Dependency Structure Matrix, DSM, where for each column representing a node, the existence of a link to another node is denoted by “1”. For instance, the column marked “Fi1” has “1” marked in rows “R1” and “Alfa”, which corresponds to the arrows denoting links between respective nodes in Fig. 5.6. A cycle in a graph appears as a coupled block in a DSM, i e if cell ij is marked with “1”, then also cell ji will be marked.

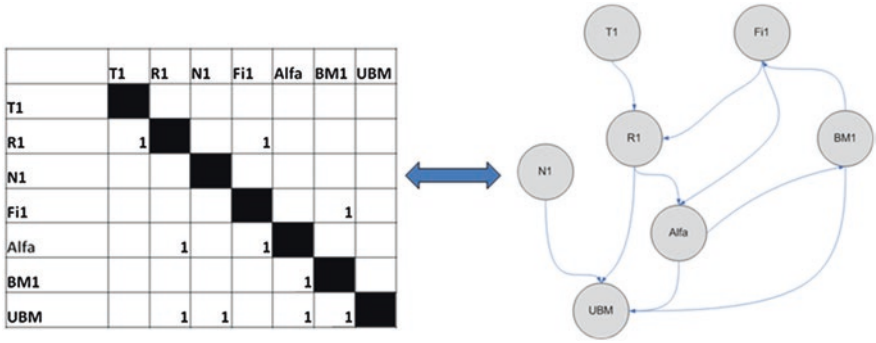


Fig. 5.6 A graph can be mapped one-to-one to an adjacency matrix

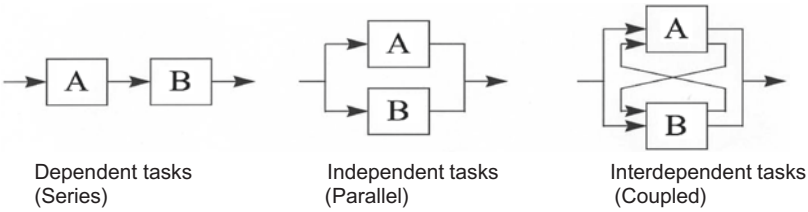


Fig. 5.7 Three types of dependencies (causalities) for design subtasks

When used for the purpose of planning complex design work (or a design automation system) it is often more convenient to set up a DSM directly rather than go via a graph. Since the two representations contain identical information the definitions and theorems developed in graph theory are fully applicable to DSMs.

In order to set up a DSM, the complete design task is divided into suitable subtasks, see Fig. 5.8. Sometimes each subtask is associated with a component in the product structure, sometimes the subtasks are of a more abstract or general nature. The subtasks label rows and columns of the matrix and are inserted in optional, but consistent, order, e.g. following an assumed sequence or the product structure. Starting from the top row of the matrix and moving downwards one should then clarify for each subtask what results from the other subtasks that are *directly needed* to be able to carry out the present task. The three alternative dependencies between tasks are given in Fig. 5.7.

Consider Fig. 5.8. Subtasks A and B require no input from other tasks, they might for instance be input parameters. They are independent and could be carried out in parallel. Task C requires input from B and B is available so C can be resolved. Task D require input from A and C, where A was available from start and C from the previous step. D can thus be resolved. Tasks C and D must be solved in sequence sin D depends on C. When we come to task E a problem arises. E depends on F which is not yet available. F however only depends on A, so if the

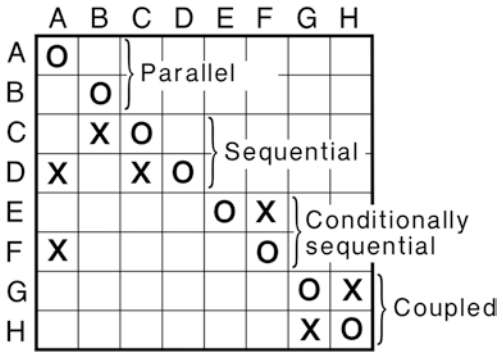


Fig. 5.8 Dependency Structure Matrix, DSM, representing eight subtasks, *A–H*, having four types of dependencies

execution order of E and F were reversed we could resolve both E and F, which are said to be conditionally sequential. Tasks G and H however are mutually dependent, coupled, and cannot be resolved one at a time but need to be treated simultaneously. Coupled subtasks often constitute the main stumbling blocks in engineering design irrespective of whether the process is to be carried out manually or by an automated system.

An example of a DSM of industrial origin is shown in Fig. 5.9. The example refers to the design of the flow channels in a thermal plate heat exchanger, see the

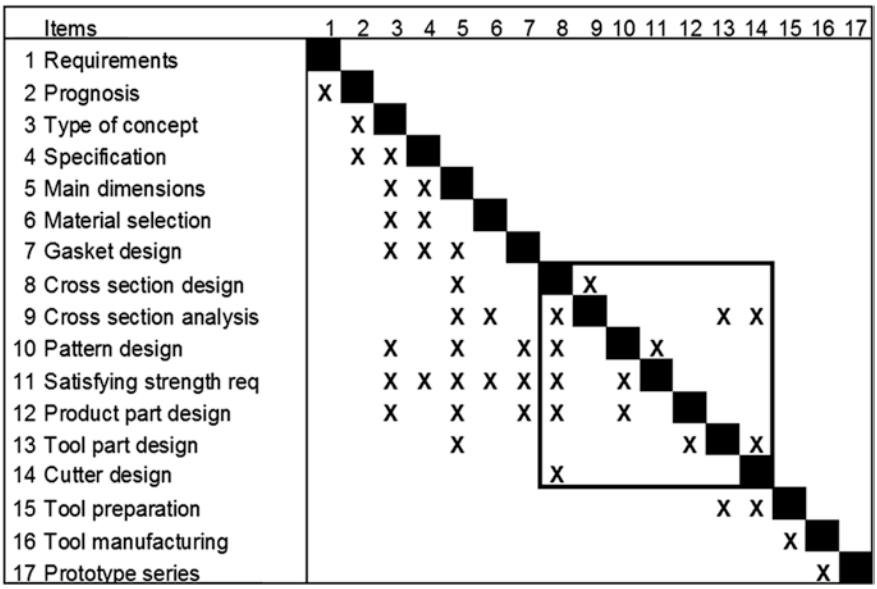


Fig. 5.9 DSM for flow channels of plate heat exchanger. The *box* constitutes an optimisation problem involving three different technical disciplines. From Rask and Sunnersjo (1998)

industrial case No. 5 in Appendix A. The DSM was used when setting up a design automation system with an objective to optimise a design solution with conflicting requirements of strength, thermal efficiency and manufacturing requirements.

5.3 Problem Structure and Choice of Solution Principles

The preceding graph theory can be used to analyze the structure of a given problem, or more precisely, the structure of its knowledge base and determine what solution principles that are applicable and suitable. In processing of explicit knowledge for design automation there are basically four alternative solution principles available:

Procedural, which represents the traditional approach to computer programming where the programmer defines the sequence of operations that are to be performed. The domain knowledge and the processing instructions are interspersed in the computer code.

Inference (or rule) based, where problem specific knowledge is stored in a knowledge base and program execution is governed by a generic "inference machine". The programmer is not required to specify the sequence of operations, only to ensure that the knowledge base is complete and consistent. As knowledge is required, the inference engine will search the knowledge base and activate the needed facts, rules and methods.

Generate and test, G&T, is a very simplistic, but robust, approach where the solution space is systematically populated and scanned for a valid solution. This method of *exhaustive search* is only applicable to problems with a finite number of possible solutions and, due to combinatorial effects, restricted to small solution spaces.

Constraint based search is a method where the set of given constraints are evaluated in order to narrow down the remaining solution space. The solution space might be so drastically reduced that only one valid solution remains or allowing the remaining, "pruned" search tree to be scanned for valid solutions using G&T.

These paradigms can also be combined into hybrid solutions. Chapter six is devoted to a more exhaustive review of these four solution principles.

The principles for mapping solution principle to problem structure can now be summarised as follows:

- A problem with data flow represented by a **topologically sorted, rooted tree** can be solved using traditional **procedural** programming techniques, as a series of assignments, branching and so on.
- A problem with a knowledge base represented by a **directed tree** can be solved using **inference** based forward or backward chaining algorithms. Allows free structuring of knowledge, which is separated from control instructions. Dynamic input data set possible only if cycles of the data flow do not arise.

- A problem represented by a search space (e.g. product structure) that is a **rooted, directed tree** and multidirectional **constraints** on **discrete** design variables can be solved by **exhaustive search** (G&T). Design variables must be discrete and finite in number. Execution time grows rapidly with the solution space, so for practical reasons only small problems can be solved using this principle. Dynamic input data set is generally allowed.
- The search tree can be reduced by applying **constraint programming** techniques (pruning). Pruning could be due to constraints that must be enforced or due to"common sense" constraints that are introduced to reduce the solution space. After pruning due to constraint processing the search tree is often reduced to a size where G&T can be applied to solve the problem.
- A problem with graphs that have **cycles** (or blocks in the DSM) must be partitioned so that the graph becomes a directed tree before solution. When the variables are real numbers partitioning is carried out using an **equation** solver or, for under-constrained problems, an **optimisation** algorithm, see Fig. 5.9.

5.4 Problem Complexity

The solution space for a given problem can, at least in theory, be estimated by applying the rules of combinatorics. The size of the solution space together with the methods used to search it determine the total number of required operations and consequently what execution time that will be required. Search based solution methods always run a risk of being slow to execute and also powerful computers may require unacceptable execution times if unsuitable algorithms are used. Of course execution time will be hardware dependent (clock time, available memory and so on), but even small changes of the problem formulation or the algorithms used might have a very significant effect upon execution time.

To estimate actual execution times in advance is a difficult task of combinatorics, but more important than absolute time is to study how sensitive solution time will be to problem size, i.e. how well the intended solution procedure *scales*. The effect of scaling up a problem can be very dramatic and it is important to be able to foresee such effects. Otherwise there is a risk to establish a design automation system that works well as a prototype, but is not viable when problem sizes grow. As scaling up is almost bound to happen for systems that come into industrial use, poor scaling properties could lead into a *cul-de-sac* that might render the system useless.

To characterise the scaling properties of a solution method a metric called *time complexity function* is used. This function is related to execution time in an indirect way and gives a qualitative indication of how suitable an algorithm is to handle realistic, large problems (scalability).

Let N represent the size of a problem, e.g. the number of design variable values. The time complexity function is then defined as

$$C(N) \sim O(f(N))$$

meaning that for large values of N, C(N) will be limited by the dominating function f(N). To give an example, consider a solution process where the number of operations have been estimated to $a + bN + cN^2$. For large values of N the third term will always dominate and hence, $f(N) = N^2$.

The following complexity classes are relevant for most common processing methods,

- linear, C(N) ~ O(N)
- polynomial, C(N) ~ $O(N^a)$, a = 2,3,.
- exponential, C(N) ~ $O(b^N)$
- factorial, C(N) ~ O(N!)

As an indication the computational complexity for the most important solution principles that will later be discussed in Chapter six are given below,

- Procedural, static sequence, C ~ O(N)
- Inference based, dynamic sequencing, C ~ $O(N^2)$
- Constraint programming + G&T C is problem dependent
- G&T, all permutations C ~ O(N!)

How then should we interpret these complexity functions? Say that we want to topologically sort a directed graph with ten nodes by G&T (brute force approach). The N nodes can be arranged in N! ways. For 10 nodes C is 3.5×10^6. This problem is solved instantly on any lap-top computer. For 25 nodes C is 1.5×10^{25}. For a computer doing one billion operations per second this will take 7.5 billion years, i.e. the problem cannot be solved irrespective of what computer power that is available. Computational methods that works correctly but result in unacceptable execution times as the problem size increases are called *intractable*. With few exceptions such methods cannot be used to solve design automation problems of realistic size.

References

Eppinger, S., Browning, T.: DSM. MIT Press, Boston (2012)

Haggarty, R.: Discrete mathematics for computing. Pearson Ed Ltd, England (2002)

Rask, I., Sunnersjo, S.: Design structure matrices for the planning of rule based engineering systems. In: Conference Proceedings Integration in Manufacturing, Esprit, Goteborg, Sweden (1998)

Steward, D.: The design structure system: A method for managing the design of complex systems. IEEE Trans. Engg. Manag. (1981)

Warfield, J.: Binary matrices in system modelling. IEEE Trans. Syst. Man Cybern. **3**, 441–449 (1973)

Chapter 6
Representation and Processing of Explicit Knowledge

Assume that we have clarified and compiled the specific domain knowledge and problem structure required to perform a certain task of variant design. This knowledge, i.e. understanding of information and methods to be used, must then be transformed into a format that can be coded as a computer program. This is not primarily a matter of conversion to specific language syntax, but rather a matter of being able to express subtle real world knowledge in the standardised formats that the computer will accept. Chapters 4 and 5 discussed the process of *idealisation,* where real world knowledge is clarified and distilled into a simplified form, while still retaining all its significant meaning and characteristics. From a representation of the knowledge in idealised form, the practical coding into a specific programming language or tool could be seen as a process of knowledge *modelling.* This is the topic of Chaps. 6 and 7.

From computer programs for *analysis* we are used to a procedural approach where a predefined sequence of instructions is executed. This is to some extent applicable also to computational *synthesis* problems, but such problems often present more divergent and dynamic solution structures. Rather than following the predefined solution paths of procedural programs, the solution paths for synthesis problems may need to be determined dynamically during execution. This implies some sort of *search* process, i.e. a computational facility to search the content of the knowledge base or the solution space for the steps that lead to a viable solution.

Representation and processing of knowledge are the key elements when planning a design automation system. Figure 6.1 gives an overview of the main factors of a computable design task. By applying the defined relations on the specified input parameters the sought after design variables will be determined step by step. The dependencies among the relations will govern how the solution process will proceed and the previously discussed DSM characteristics provide decisive criteria

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_6

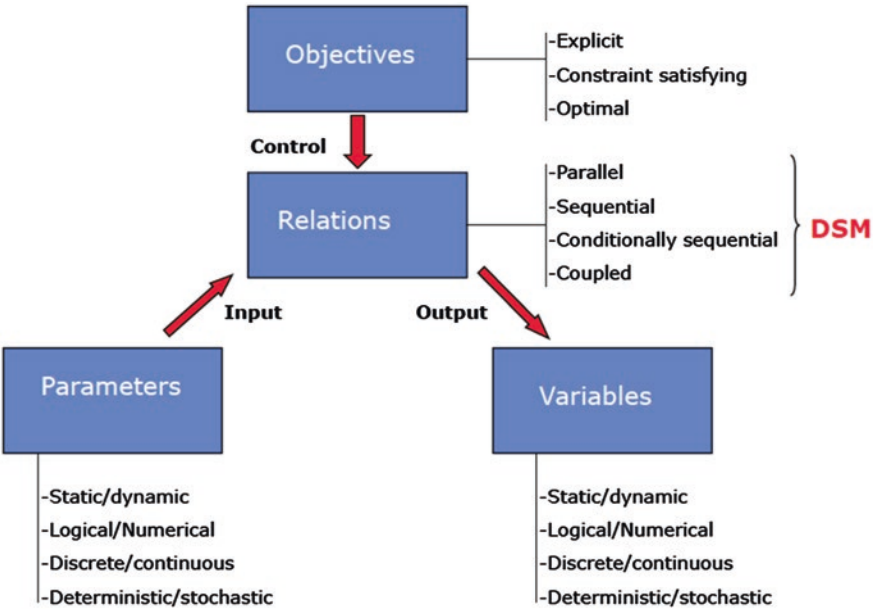


Fig. 6.1 Main factors governing the mode of operation of a design automation system

for what methods that can be used. The overall objectives determine when the search process is finished. Hence objectives

- "Explicit", implies that one and only one solution is to be found
- "Constraint satisfying", implies that several viable solutions exist, but the first solution encountered is accepted
- "Optimal", implies that the best possible solution is required

The nature of the technical knowledge required to solve typical tasks in engineering design was discussed in Chap. 4. With these discussions in mind we will now review how the available programming methods match these categories. The multidisciplinary nature of design work indicates that in many cases one, single computer method will not suffice, instead a combination is necessary. Also it is very likely that for engineering applications, the knowledge will have elements that require logic reasoning as well as numerical computations in the same program.

In this text the characteristics and origin of the knowledge used have been the decisive element when categorising the methods described for design automation applications. One watershed thus, is whether design rules are, or can be made, *explicit* or whether they are *implicit*, i.e. imbedded in knowledge related to the problem. The former type of methods are covered in the present Chap. 6, while the latter type will be discussed in Chap. 7. Referring to Fig. 1.4, this means that systems based on the declarative paradigm and systems based on traditional procedural programming are treated here while systems categorised as Computational Intelligence are treated in Chap. 7.

Procedural programs are fast and efficient for problems with heavy numerical computations and where the problem structure is hierarchical. In procedural programs the "knowledge base" is mixed with process control instruction, a fact that is unfavourable from the perspective of overview and quality assurance of the program. Programs written according to the declarative paradigm differ significantly from the procedural programs by a strict division between the *knowledge base* and the solver, which is called *inference engine.* The knowledge base is strictly aimed at representing the knowledge governing the problem without consideration to its processing. The design rules may be stored in a way that is beneficial to the user and gives a good overview.

If the data flow graph for a problem has cycles, the inference based systems run into trouble by getting stuck in eternal loops. Instead methods that search the solution space can be applied. The basic, straightforward way of Generate and Test is limited to small problems, but in combination with a constraint solver also full scale industrial problems can be solved. All these methods have one thing in common: The knowledge base consists of elements that give an explicit definition of the knowledge that is to be used to search for and, if possible, find a design solution that satisfy given requirements and constraints.

6.1 Procedural Solution Methods

Traditional computer code works according to the procedural or imperative principle. The program consists of a list of instructions that the computer will perform one at a time and in sequence. Before approaching the actual coding it is up to the programmer to plan this sequence of operations (input/output, file storage, calculations, loops, jumps, sorting…) and to anticipate at which points there is a need to introduce branching of the execution process. A detailed scheme needs to be worked out which will serve as input to the task of writing the program.

In the resulting code instructions related to the domain specific design knowledge will be intertwined with instructions related to execution control. The "knowledge base" is thus dispersed over the entire program code and structured according to what is needed during the execution process. This is a situation that arises by necessity in procedural programming. It has its advantages but in the context of design automation also some significant disadvantages. It will be difficult to get an overall view of the accumulated domain knowledge and thus difficult to control the relevance and accuracy of the knowledge that the program uses. Further, since the knowledge is not structured in any systematic fashion but only according to the execution process, it might be difficult to localise a specific knowledge element that might need control or updating. Programs for industrial applications are often developed in steps over long time and by many programmers and becomes gradually more and more difficult to maintain, one reason being that a required change in the sequence of instructions may cause side effects to other parts of the program that are difficult to foresee.

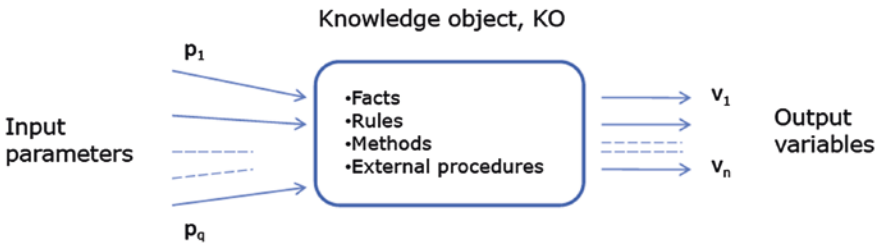


Fig. 6.2 Knowledge object as a black box system further elaborated in Section 6.2.4

One way to at least partly separate knowledge and execution control also for procedural programs is to set up a work-flow manager separate from the executable statements representing the domain knowledge, see the bulkhead design example No. 6 in Appendix A. In this application an executable design manual has been set up consisting of knowledge objects (see Fig. 6.2) for all subtasks of the design calculations. The required chain of operations are then defined in a workflow manager graphically represented by a DSM.

This arrangement gives clarity and overview but the workflow is static, which means that if any changes are required it is up to the programmer to redefine the work-flow taking into account any side effects these changes might have. The radical solution to the problem is to turn to programs using an *inference engine*. Such programs will allow a separate knowledge base that could be arranged for clarity and overview rather than execution control. If the problem structure is such that the information flow graph has cycles, i.e. cannot be transformed into a topologically sorted tree, other methods are required.

Having said this in favour of inference based systems it must also be said that traditional, procedural programming methods still have, and should have, a significant role in many design automation systems. In cases with few variations of solution sequence or where the execution needs to be directly controlled, procedural programs may well be the best choice. One important case where execution order must be predefined is when product geometry is to be built up under program control in a history based CAD system, see Sect. 3.1.1. Also, it is very common that methods used as parts of the knowledge or constraint based systems that will be discussed later, are written using procedural programming methods. Referring to the previously discussed graph theory, procedural programming is applicable, and often preferable, when the data flow is hierarchical as in a topologically sorted tree or in a DSM with all marks under the diagonal.

6.1.1 Design Problems Cast in a Mathematical Form

Procedural programming is often the natural choice for computable design problems that can be cast in a mathematical form. Assume that a design solution is defined by a set of N real design variables, x_i; i = 1, … N. Out of the N design

variables Q are given as input parameters. The design variables are constrained by the following M relations:

$$\begin{cases} f_1(x_1; x_2; \ldots x_N) = 0 \\ f_2(x_1; x_2; \ldots x_N) = 0 \\ \ldots \\ f_M(x_1; x_2; \ldots x_N) = 0 \end{cases}$$

For this formulation of a design problem the following conditions determine what solutions one would expect to find:

- If N − Q = M, the problem is *fully constrained* and at the most one solution for x_i exists,
- If N − Q > M, the problem is *under-constrained* and many solutions for x_i could exist,
- If N − Q < M, the problem is *over-constrained*. In the general case, no solution for x_i exist

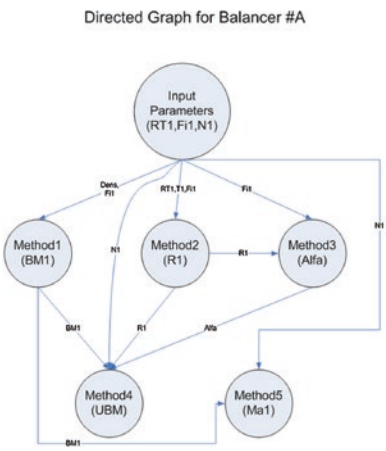
The solution procedure is independent of which x_i that define input parameters and design variables respectively.

There certainly exist design problems that can be formulated in this way and the solution procedure is then comparatively straightforward. In engineering praxis however there are often complications. The design variables might be a mixture of real, integer and logical variables. This means that equations are replaced by bounds and intervals or logical operations. Further, design problems are often under-constrained, which leads to a constraint satisfaction or an optimisation problem rather than a solution of simultaneous equations.

A more flexible solution approach is therefore often called for. Rather than solving a system of equations it is natural to define a solution sequence of operations, simple or more complex, representing a fixed workflow. This is a viable approach if the sequence remains the same or only has a small number of variations, which could be implemented as conditional branching. Let us return to the example with the dynamic balancer described in Sect. 5.1 and Fig. 5.2 and investigate how a procedural program would work, and what problems that might be encountered.

6.1.2 Case of Application of Procedural Solution Method

Say that the task is to predict the unbalance capacity for a given balancer design. The information flow of the calculation is given as a graph in Fig. 6.3, left, and the corresponding DSM, right. The nodes of the graph are the methods (formulas) used to calculate respective variables (indicated in brackets), while the links between nodes transfer the variables. Correspondingly the DSM has the subtasks, named after the variable computed, inserted as row labels. The details of the



Balancer design problem #1		1	2	3	4	5	6	7	8	9	10
RT1=	1	1									
Density=	2		2								
T1=	3			3							
N1=	4				4						
Fi1=	5					5					
BM1	6		X			X	6				
R1	7	X		X		X		7			
Alfa	8					X		X	8		
UBM	9				X		X	X	X	9	
Ma1	10				X		X				10

Fig. 6.3 Graph and DSM illustrating information flow for calculation of capacity of automatic balancer

methods required are not essential from a design automation perspective and are thus given in Fig. 6.4 without explanations.

We can see that the graph of Fig. 6.3 is a rooted, topologically sorted tree and, correspondingly, the DSM has all marks below the diagonal showing a hierarchical information flow. According to the criteria of Sect. 5.3, it is clear that the problem can be solved by a procedural program that will return a value for unbalance capacity and a value for total weight of balls.

Balancer design problem #1		1	2	3	4	5	6	7	8	9	10	Relations
RT1=	1	1										200
Density=	2		2									7800
T1=	3			3								4
N1=	4				4							8
Fi1=	5					5						12
BM1=	6		X			X	6					$\pi \cdot \Phi_1^3 \cdot \rho \, 1/6$
R1=	7	X		X		X		7				$RT_1 - T_1 - \Phi_1/2$
Alfa=	8					X		X	8			$\Phi_1/(2R_1)$
UBM=	9				X		X	X	X	9		$2BM_1R_1 \sum_{BNo=1}^{N_1/2} \cos(\alpha(2BNo - 1))$
Ma1=	10				X		X				10	$N_1 BM_1$

Input parameters

Intermediate variables

Design property variables

Fig. 6.4 DSM for balancer with methods for each subtask specified showing dependencies between input parameters, intermediate variables and design properties

6.2 Inference Based Systems

An industrial production system for design automation is often a software package of considerable size, e.g. the system described in case 2 in Appendix A contains about 25,000 design rules. To sort out and keep track of the solution process correctly is not a trivial task, especially after a couple of years of use. By then many individuals have over time contributed to the knowledge base and it is obvious that quality assurance and overview become serious issues for such systems. Also safe and easy updating of the system is of great importance. To have the knowledge base merged into a procedural code and intertwined with control instructions is a significant difficulty in these respects.

In many cases it would be better to state only the design rules and then have the computer find the right path through the knowledge base and process the rules. The inference based method provides such an alternative, which is applicable to problems that have a hierarchical tree structure but without the need for topological sorting. Contrary to what is the case for procedural systems, inference based systems operate without a predefined solution path, which is instead generated during execution and governed by input parameters and intermediate results. Among such results might well be human interaction in case the system demands complementary information from the user.

6.2.1 Separating Knowledge and Control

The fundamental concept behind inference based solution principles is to separate the program into two parts:

- The **knowledge base** that contains all the domain specific knowledge that is required to arrive at a solution. The knowledge base is set up by the user, who does not need to be concerned with how the knowledge should be applied. Hence, the order in which the design rules are stored is unimportant but it is the user´s responsibility that they are correct, consistent and complete. This method of programming is called declarative.
- The **inference engine** is generic and controls how the knowledge base is executed at run-time. The inference engine is not related to any specific problem and operates outside the control of the user. It is often written in procedural code and has as its main task to scan the knowledge base to clarify which design rules that should be applied and in which order.

The knowledge base in an inference based system is made up by:

- **Facts**, which could be *given facts*, e.g. data given as input parameters or permanently defined in the computer code when programmed. It could be *transient facts* supplied by the user or sensors during execution, or it could be *derived facts* that are determined by the rules and methods of the system during execution. *Examples*: *Planet is Earth. G has the value 9.81 m/s*^{2}

- **Rules** (or *Production rules*), which are conditional statements of the form "*if* (condition) *then* (action)" where a predefined action is performed when a certain condition is met. Conditional statements may be combined using Boolean operations like "AND, OR, EQUAL" *Example*: *if* (*Planet is Earth*) *then* (*a is assigned the value G*)
- **Methods,** instruction(s) on how to determine facts that are requested when needed. *Example*: *Perform calculation* $at^2/2$ *and assign the result to S*

The vocabulary concerning the term "rule" is somewhat confusing. What is here called "inference based systems" is often referred to as "rule based systems". Rule then stands for design rules, consisting of "facts, production rules and methods" not restricted to the conditional production rules only.

We will now illustrate the search process with a very simple example. Assume that we want to determine the distance *S* that a body will fall when dropped on planet *earth* during time *t*. To solve the problem we could apply the facts, rules and methods given above. The path of search and reasoning would be something like this: *S* is the sought unknown variable and we can see that a value is assigned to *S* by the formula in the **method**. Time is assumed to be given input, but acceleration, *a*, is unknown, so we must first search for a rule that assigns a value for *a*. The action part of the **rule** will set the value of *a* to *G* providing the rule condition is "true". Planet is specified as *earth* so the assignment applies. The numerical value for *G* is found as a **fact** and the formula defined as a method can now be evaluated and *S* assigned a numerical value. Unlike in a procedural solution sequence the order in which the knowledge is defined is irrelevant—the problem is solved by a search process that we will later recognise as a process of "back-ward chaining".

The three elements of the knowledge base represent formats, which in combination go a long way towards capturing the design knowledge required to perform a given design process. Given facts and methods are static elements which are called when required, while the actions of rules are dynamically determined during execution and depend on the current values of variables that form the conditional part of the rule. The three elements are often used in combination, e.g. a fact is utilised as part of a condition in a rule or the action in a rule triggers a method to determine the required result. A method might very well be a more complicated procedure than an algebraic formula. It could be a call to an external program performing numerical computations, interpolation, selection in a data base and so forth. Figure 6.5 illustrates how the inference engine step by step derives new facts from the given input parameters until eventually all required design variables have been determined.

In its simplest form the inference engine is a program that scans the knowledge base to search for opportunities to determine the requested design variables by execution of the rules with fulfilled conditions. Figure 6.6 illustrates how the conditions of the N_max number of rules in the knowledge base are tested. The rules found to have satisfied conditions are "triggered" and stored in the "conflict set". One of the rules is selected (many different strategies for selection exists)

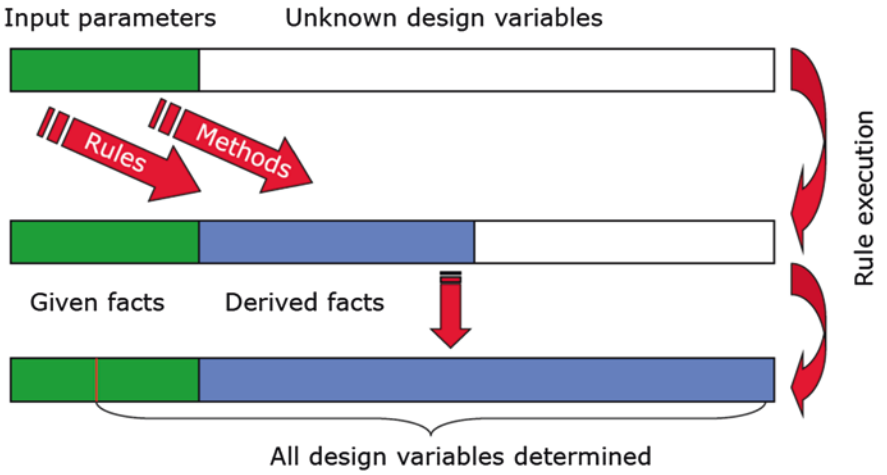


Fig. 6.5 The requested design variables are made up of given input parameters and derived facts determined in an iterative procedure

and the action is "fired". The resulting new, derived fact(s) is/are added to the current fact base and the scanning is repeated with the new fact included. This process is repeated until no more rules with satisfied conditions remain. The final fact base will now contain all requested design variables provided the knowledge base is complete and consistent and the rules do not have dependencies which form cycles.

The process described starts with the given facts, applies iteratively the design rules and methods and then determines all design variables that will result. This process is called *forward chaining* and one characteristic is that all results will be derived irrespective of what was really sought for. An alternative approach was described in the previous example where a distance S was sought. This process works backwards from each sought variable and derives how it is related to the given facts. This process is called *backward chaining* and is the more efficient method when only parts of the design variables are required.

The described process is very basic and would be slow to execute due to its iterative character. Commercial software packages use various approaches to improve efficiency. One obvious waste of time is to evaluate the conditions of the rules each time the loop in Fig. 6.6 is executed. More sophisticated system use algorithms that save the test results to the following iterations e.g. the Rete algorithm (Coppin 2004). For the user however, in most cases it is sufficient to understand the basic principle and the fact that the more sophisticated algorithms, will return the same results, only much faster.

For both backward and forward chaining in their simplest form, the knowledge base is searched for applicable rules. These could be stored in a rooted tree structure, e.g. a product structure is a common and natural storage principle. When searching this tree the search could follow each branch to the leaves of the tree,

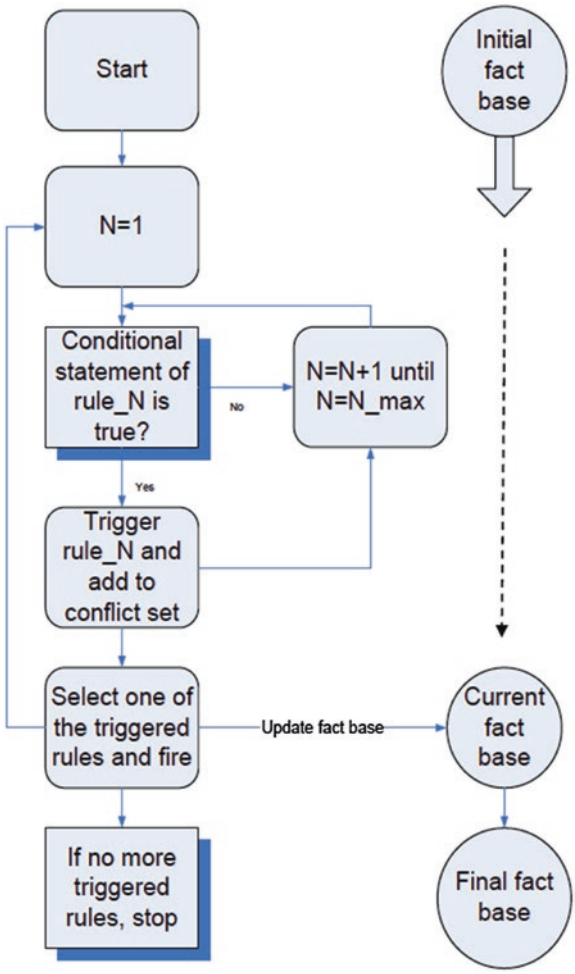


Fig. 6.6 Fundamental flowchart for basic inference engine

one at a time. This is called searching *depth first*, while the alternative *breadth first* implies that of the breadth of the tree is visited first and then step by step moves down into the tree.

6.2.2 *How Is Inference Based Programs Different from Procedural Programs?*

The inference based system works autonomously and creates a solution path as it processes the design rules in the knowledge base. This mode of operation is very different from what we are used to in traditional programs. Let us now return to the balancer example having a workflow for procedural programming shown in Fig. 6.4.

The left column lists the parameters and variables, while right column contains the relations that should be enforced. In the figure the parameters and variables are topologically sorted, i.e. all indications in the DSM appear below the diagonal. This would not have been necessary if a rule based system had been used which allow the relations to be stored arbitrarily and the solution path found during execution.

So what is gained by use of this programming paradigm? Obviously the answer is dependent on problem characteristics and the particular software used, but generally speaking it can be said that inference based systems have:

- Modularised program architecture, i.e. facts, methods and rules can be added, altered or removed without having to consider any side effects on the execution process. The execution of rules is demand driven and determined during execution and it is dynamic and adapts to changes of the knowledge base,
- In principle, no need for control logic,
- Simpler and shorter code. No "Spaghetti code", which can be problematic in procedural code if the knowledge base contains a large number of complicated, nested conditions.
- Better management of design rules. The fact that the path of execution is not (critically) dependent on the order in which the design rules are stored provides the opportunity to store them systematically for easy overview

Are there any disadvantages?

- Facts, methods and rule executed entirely governed by an inference engine is an idealised situation. In engineering design it is sometimes necessary to enforce a certain sequence and to override the inference engine with a predefined execution sequence. One example is when a CAD model with a history tree is generated, which cannot be done in an arbitrary order. A feature to determine the order of execution is therefore included in some software packages for inference based engineering systems
- There is a risk of long execution times for large knowledge bases
- Since the path of execution is dynamic and therefore not known by the programmer beforehand, debugging might be more difficult than in procedural systems. Also, it is essential to understand how the inference engine works in order to write the rules for correct processing
- It might be self-evident, but still needs saying: Irrespective of what programming method that is used, it is still up to the user to verify completeness, consistency and correctness of the knowledge base and that the rules do not have cyclic dependencies.

6.2.3 Frames for Object Oriented Knowledge Base

An important advantage of rule based programs is that the elements of the knowledge base can be stored in any order that is practical. This provides an opportunity to arrange the knowledge base in a tree structure for efficiency and ease of

Frame: **Truck**	Authour: Staffan
Superclass: Vehicle	Subclass: Construction
Slots	**Facets**
Colour	Blue
Engine	V6, V8
Wheels	if (weight>10 tonnes) then (No_wheels=6) else (No_wheels=4)
Price	Price=call external method

Fig. 6.7 Template for Frame or Knowledge object

management. The paradigm of object oriented programming is realised in the AI world by the concept of *frames*.

Frames, which could be seen as front-ends of rules and methods, allow knowledge to be represented in hierarchic structures with classes and relations. This approach implies:

- *Knowledge abstraction*: Divide knowledge base in classes with relations (e.g. "part-of", "consist-of") to improve structure and overview. Generalisation (up) and specialisation (down)
- *Inheritance*: Facts, rules and methods in one class are passed on to frames in all sub-classes
- *Encapsulation*: Each frame acts as a black box processing input data (facts) and delivering output data (updated facts).

Figure 6.7 shows an example of the template for a Frame. This will store:

- Meta data: Name, author, date…
- Place in hierarchy: Superclass (parent), subclasses (children)
- Slots: Attribute classes
- Facets: Contents of slots. Facets could be e.g. default values, facts, rules, methods, calls to external procedures.

6.2.4 Knowledge Objects

These can be seen as an extension of Frames, for cases where the subtasks require sizeable programs and might also include external programs written in other languages. The underlying idea behind knowledge objects is to divide the design

knowledge into suitable "chunks" that behave as black boxes with only input and output values available outside the object, see also (Sunnersjo et al. 2006). The work-flow manager or inference engine then searches for objects where all input parameters are available, executes that object and adds the resulting variables to the current facts base and starts a new search, but now with the results from the previous loop added. This process goes on until no more objects can be executed.

One can visualise the organisation of the system as a class room with the students in front of a black-board divided into cells where the current values of all design variables are written, Fig. 6.8. The system is *event-driven*, e.g. action is initiated when a value is changed and represented by students having different assignments that are initiated when the values of one or more cells drawn on the blackboard are changed.

In the example the process starts when the value in cell 22 is changed by outside intervention from 1 to 5. Imagine that KO3 has been told to monitor cells 11 and 22 and act so that rule KO3 is always enforced. Since the condition of this rule is true after the change, this student will now change the colour of cell 23 to blue. This in turn alerts KO2 who will, according to rule KO2, add the number one to the previous value of cell 12. Processing then halts until some new event initiates further processing.

The knowledge objects could be substantial programs, often of a procedural nature, that perform operations in isolation from the calling program and each other. Operations can range from calculation of an algebraic formula to data base searches, interpolation or optimisation. It could also involve calls to external programs, like geometry manipulation using a CAD system, structural simulation using Finite element software or more specialised technical computations such

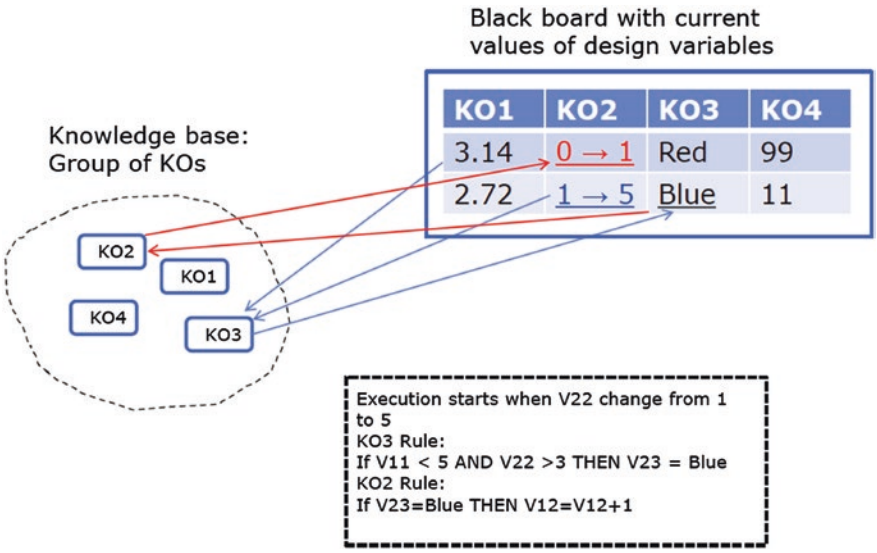


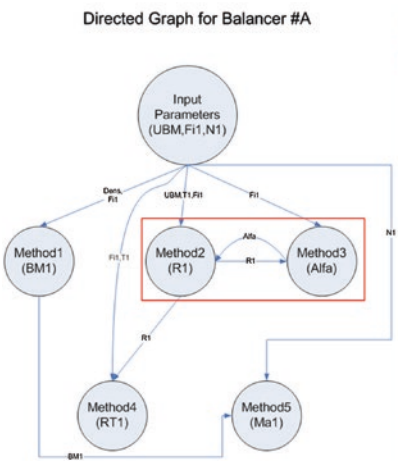
Fig. 6.8 Inference based system with knowledge objects seen as a black board analogy

as property calculations of performance predictions. The organisation of a design system according to this architecture provides a very high degree of modularity and flexibility. New objects can be added or old objects replaced with object based on new technology without affecting other parts of the system. The inference engine then acts like a manager that initiates, controls and evaluates the results generated outside the rule system. Example No six in Appendix A is built using knowledge objects governed by a static work-flow manager, while example No nine, involving heavy numerical simulations, uses an event driven process control.

6.3 Exhaustive Search by G&T

6.3.1 Problems with Discrete Variables and Cyclic Flow Graphs

We now return to our example of the balancer in Fig. 5.2. It was pointed out in the previous section that by using an inference based system we are free to store the relations in any order we like without affecting the result. But suppose we want to reverse the task compared to previous example, e.g. define a required unbalance capacity instead and ask what size of balancer outer radius that is required, all other variables remaining constant. Clearly this problem requires the same general product knowledge and design rules as previously but this program will not solve the problem directly. Instead the graphic representation of problem structure in Fig. 6.9, shows a cycle between the two variables R1 and Alfa. This is equivalent to the coupled box in the DSM.



Balancer design problem #1		1	2	3	4	5	6	7	8	9	10
UBM=	1	1									
Density=	2		2								
T1=	3			3							
N1=	4				4						
Fi1=	5					5					
BM1=	6		X			X	6				
R1=	7	X			X		X	9	X		
Alfa=	8					X		X	8		
RT1=	9			X		X		X		7	
Ma1=	10				X		X				10

Fig. 6.9 Data flow graph with one cycle and DSM with coupled block

As the problem is formulated it cannot be solved directly in the form of an explicit answer resulting from a chain of assignments. The two methods of the cycle must be merged into one in order to make the information flow of the problem hierarchic, an operation called *partitioning*, see Fig. 6.10. The merger is in this case done using a solver for a system of equations, which becomes the method that replaces the original two methods denoted two and three in the Figures.

After partitioning it appears that the task could be solved in a straightforward way either by a inference based program or procedurally using a short sequence of assignments. There are however two significant drawbacks with this approach, that explains why other methods are often preferred in a case like this.

First, the variables will often be a mixture of real and integer numbers. The balancer problem is represented by design parameters, intermediate variables, design variables and product properties. These will be real numbers (e.g. weight) or discrete numbers, either by necessity (e.g. No of balls) or choice (e.g. diameter of balls). It is common that the main design variables that define the product are discrete numbers.

For the current example, the equation solver will in general return a real value for ball race diameter, FI1, but for reasons of standardisation it might only be acceptable with solutions having integer variable values. Simply rounding to nearest integer value would introduce an arbitrariness that is not acceptable. But constraining FI1 to discrete numbers means that the equalities will generally not

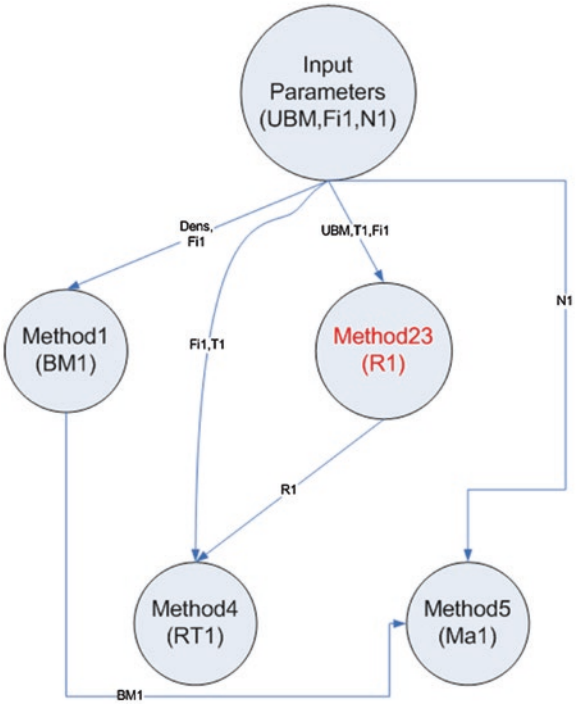


Fig. 6.10 A coupled problem reduced to a hierarchic tree structure by partitioning

be satisfied. Instead we need to introduce inequalities (bounds and intervals). In the balancer case we could replace the constraint relating to UnBalance Moment, UBM = 6.0 Nm with the inequality UBM ≥ 6.0 Nm, which would result in a balancer that has a balance capacity equal to or better than the original requirement. This mixture of real and discrete numbers is a stumbling block that is not easily solved with the straightforward procedural or rule based solution.

The two processing sequences of Figs. 6.4 and 6.9 respectively, return numerical values to two required design variables. In practice there are many more sets of given parameters and requested output variables that the system must be able to handle to be a useful tool for adapting the basic balancer to varying customer demands. One customer might want to minimise the ball diameters to reduce the thickness of the device, one will only allow the same ball diameters in all three tracks, one might allow one track only and so on. For the balancer at least 15 different combinations of input parameters and corresponding design variables could be expected in practice. All computations are based on the same design rules but there is a need to reverse and rearrange the relations.

If we apply procedural methods, one solution procedure needs to be programmed for each planned input data set. An overwhelming task if the number of possible variations of input data is large. If an inference based solver is used the solution sequence is determined at run-time and depend on the specified input data set. For this to be straightforward the corresponding Graph must be free from cycles and non-directed, i.e. the relations reversible. Otherwise, as is the case for the balancer, the DSM needs to be partitioned and a solution method for each partition needs to be constructed. This is clearly not an easy task for an automatic system.

To overcome these problems a seemingly very primitive method will now be introduced.

6.3.2 Generate and Test, G&T

Consider the system of equations in Fig. 6.11. To solve this we would normally substitute the expression for y of equation number one into equation number two and solve using the well-known formula for quadratic equations. An alternative, brute force approach could be to simply try a sequence of values for x and y and test against the two given constraints. In the example we find that the combination of x = 2 and y = 4 satisfy both constraints. This might appear as a very primitive way of solving a mathematical problem, but there are problems where this method of *exhaustive search* has attractions.

Generally speaking, exhaustive search suffers from poor scalability. Execution time will grow exponentially with problem size and the method is what we in Sect. 5.4 called *intractable*, i.e. problems with large solution space will cause a combinatorial explosion. For this method to be applicable two conditions must

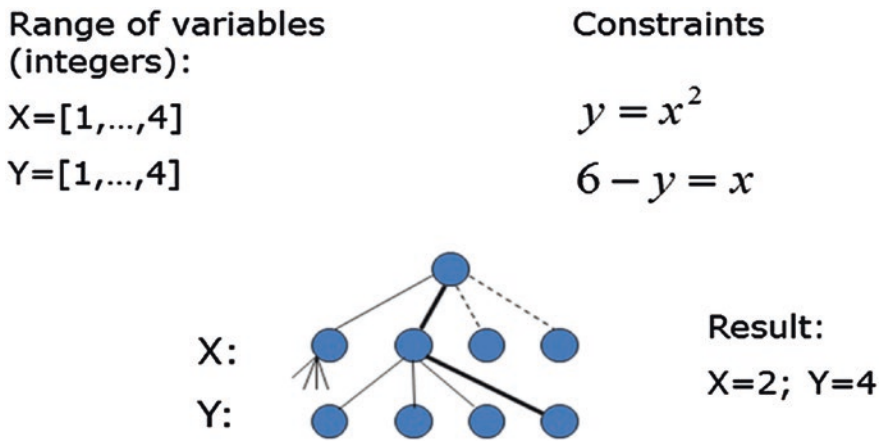


Fig. 6.11 Search space with 16 trial solutions for the system of equations

be met. The search space (rooted tree) must be finite, i.e. the nodes must represent discrete events such as the design variables being restricted to integer values. Secondly, the allowable variable ranges must be possible to estimate and this estimate used to keep down the size of the search space.

For large problems G&T is not a practical alternative. The reasons why this brute force approach could play a role despite its poor scalability is that design variables such as main dimensions, part selections, number of components and so forth, naturally are defined as integer values anyway. Further, for variant design the allowable ranges of design variables are normally well known from experience and can confidently be specified within reasonably tight limits.

We now return to the example with the balancer, this time a version with only one track. For this example a minimum value for the unbalance moment, UBM, is required and the task is to determine the three main design variables so that a design is found where the UBM requirement is satisfied. The problem could be defined as a constraint satisfaction problem:
Main design variables:

- Ball diameter, Fi1. Range: 8–17 mm in steps of 1 mm
- Disc radius, RT1. Range: 200–249 mm in steps of 1 mm
- Number of balls, N1. Range: 2–20 in steps of two

Property variables:

- Total mass of ball set, M
- Resulting UnBalance Moment, UBM

Objective:

- Find a combination of Fi1, RT1 and N1 that satisfy the specified constraint, UBM > 6.0 Nm.

The solver will scan through the tree of combinations illustrated in Fig. 6.12 until the first satisfactory solution is found. As mentioned before the search method could be either breadth-first or depth-first or something more sophisticated, e.g. search starts at a point where previous experience has shown that many satisfactory solutions exist. Note that the answer will be different depending on how the tree is arranged and how the search is carried out. All we will know is that the resulting solution satisfies the given constraints of the problem.

However, often constraint satisfaction is not sufficient, one wants to find the best possible solution, which in our case could be interpreted as finding a design that satisfies the requirement on unbalance moment with a ball set weighing as little as possible. The objective is then changed to:

- Find a combination of Fi1, RT1 and N1 that minimises (M) subject to the constraint UBM > UBM_specified

The solution space for the problem is shown in Fig. 6.13. Using the G&T method all possible variable combinations are created and their respective properties are calculated and stored. Then, in a second step, these solutions and their properties are tested one by one against specified constraints until a satisfactory solution is found. Note that there is no way of knowing beforehand whether one, many or no solution exists. If one, and only one, solutions is found, this is of course the

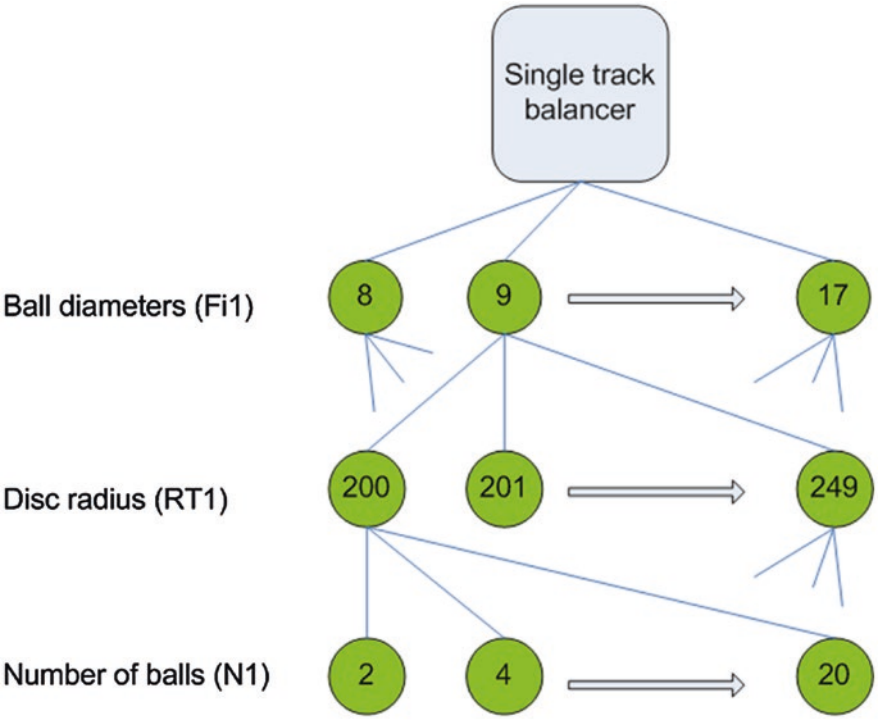


Fig. 6.12 Solution space for single track balancer

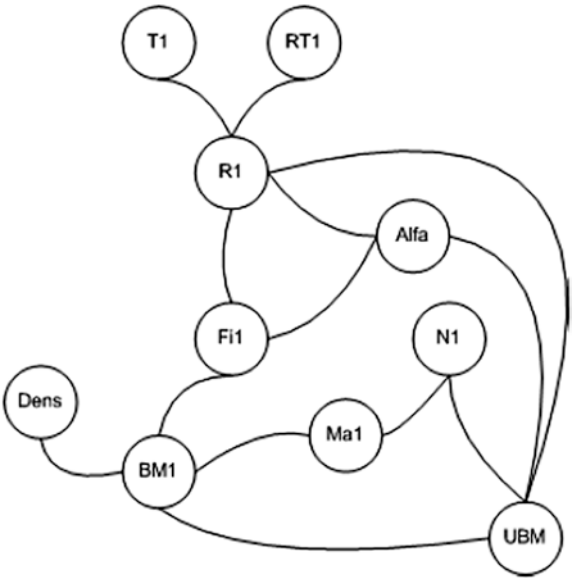


Fig. 6.13 Constraint network for balancer with one track

answer. If no solution is found we need to either relax the constraint specifying UBM or extend the ranges for the design variables. If we find more than one satisfactory solution, these will be sorted with respect to the value of M, total mass of ball set, and the solution with the lowest value of M is selected.

The solution space is calculated from the three sets of main dimensions as $10 * 50 * 10 = 5000$ trial solutions. The calculation and testing of this solutions space is almost instantaneous on a modern PC. If we add a second track there will be half a million trial solutions, still allowing acceptable solution times. When we increase the number of tracks to three or four, execution time will however become a problem, in particular if more variables are added. The practical usefulness of G&T is therefore limited, but it is important to understand that constraint processing, which will be presented below and is much more efficient but also more complicated, will appear to work in exactly the same way when seen from a user perspective.

6.4 Constraint Processing

As we have seen G&T is a versatile method that works also for cyclic data flow problems and where the solution sequence is reversed. As the number of variables and their respective ranges increase however, the search space quickly becomes excessively large. A more sophisticated approach is then required which combines G&T with a technique called *constraint programming*. The purpose of using a constraint solver is to remove unfeasible parts of the solution space ("pruning the search tree") and thus reduce the problem size before applying G&T.

6.4.1 The Method of Elimination

From a user perspective constraint based systems resemble inference based systems in that the knowledge is declarative and the control of execution is left to a generic constraint solver. The *knowledge base* of the inference system corresponds to a *constraint store* and the directed dependency graph is replaced by a nondirectional *constraint network*, see Fig. 6.13. Similarly to the rules in the knowledge base, the constraints in the constraint store can be arithmetic, symbolic, conditional, combinatorial and make calls to external programs. The order in which constraints are specified has effect on the results but not in a critical way (only the order in which solutions are found) and consequently constraints can be added or removed without considering side effects. Unlike the rule systems, constraints have no causality (direction), meaning e.g. that $x < y$ infers that $y > x$. There is normally a strong interaction between the constraints which appears as variables being shared by several constraints.

Constraint based methods are applicable only to problems with discrete variables and a finite solutions space.[1] The fundamental principle behind these systems is the *method of elimination*, i.e. the design knowledge defines what parts of the solution space that are invalid and removes from the initial solution space those variable values that cannot be assigned. What remains of the variable sets are by definition valid solutions and these solutions can be generated and tested in search for the best combination of properties.

The workflow when processing constraints follow the steps below:

1. Define the Constraint Satisfaction Problem, CSP, in Fig. 6.14 as **S(X, D, C**) where:
   - **X** = (x_1; x_2; … x_N) are finite sets of design variables
   - **D** is a finite set of domains in which the variables are ordered
   - **C** is a set of constraints
   - Example: x_1 = (**D1**) = (4,6,8); x_2 = (**D2**) = (6,8), i.e. the solution space comprises six solutions.
2. Enforce constraints by activating the stored constraints
   - Example: **C1**(x_1): $x_1 \neq 6$; **C2**(x_1; x_2): $x_1 < x_2$;
3. As constraints are applied parts of the initial solution space in the form of variable domains are removed (e.g. truncation of lists, substitutions…).
   - Example: Due to **C1** the value of x_1 is not 6 and due to **C2** it is not 8, thus it is always 4. Due to **C2** x_2 is not 4, it must thus be either 6 or 8. Hence the solution is **S** = (x_1; x_2) = (4,6);(4,8)

[1] Some commercial systems have developed methods to handle also real number variables.

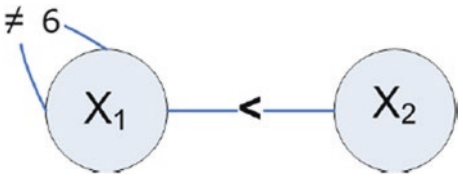


Fig. 6.14 Constraint graph for example above

4. When no more reduction of the solution space is possible, an exhaustive search of product properties for the remaining valid variable combinations is done, the solution properties are evaluated and the best solution selected.

6.4.2 Constraint Processing in Practice

When defining a constraint satisfaction problem it is not possible to know whether the problem has one, many or no solutions. What course of action is available under these circumstances?

- The cases where one and only one solution exists presents no problem
- No combination of variables within given ranges satisfy constraints: The problem has no solution. A possible course of action is to review all constraints and relax if possible. Possibly an increase of variable ranges could remedy the situation.
- Several combinations satisfy all constraints: If constraint satisfaction is enough, terminate search when the first solution is found
- Several combinations satisfy all constraints but only the best solution i acceptable. All solutions are determined and either of the two alternatives below are chosen,

  – Define objective function, e.g. cost, evaluate the objective function for all solutions and pick the best
  – In iterative steps, tighten constraints (e.g. cost, weight) until only one solution remains (narrowing of search space)

Efficient pruning of the search tree is core computational technology of commercial constraint based systems and is a science in its own right. From a user perspective it can be assumed that the system works as is if it did actually evaluate all possible combinations and then discarded those that violate given constraints while retaining the acceptable solutions. For reasons of efficiency the systems do not work in this simplified way, instead different heuristics are used to reduce the search tree. From a user point of view however, the concept of exhaustive search is a perfectly acceptable mental model to use for planning.

To get a glimpse of how efficient pruning is achieved a few methods often built into constraint solvers will be very briefly summarised. All the methods are based on the concept of enforcing the given constraints while retaining full consistency

of the defined problem. Application of a constraint on one variable will often affect also other variables and these effects are enforced by use of *propagation rules*. Below are examples of the most basic approaches intended to remove unfeasible parts of the search tree:

- Reduce constraint set to unary (one variable) and binary (two variables) constraint
- Enforce *node consistency* and remove unary constraints (reduce initial ranges)
- Enforce *link consistency* by evaluation of all links and removal of domain elements that are inconsistent. Note that as one domain is revised, the procedure has to start all over again which is very time consuming! Much more advanced algorithms exist and are employed in commercial systems.
- *Path consistency* is checked and enforced where three or more nodes are linked together. Straightforward application is very inefficient, but more advanced algorithms exist.

Irrespective of method the intention is to "prune" the search tree by drawing conclusions from the constraint set about which branches that are feasible and which branches that are not and consequently should be removed from the search tree. The constraints emanate from the relationships in the problem which could relate to some physical phenomena. For the example with the dynamic balancer one such constraint was the required centrifugal force. It is however often possible to introduce constraints of a more heuristic nature to reduce the problem size, and it is up to the user to formulate such "common sense" constraints that are effective in reducing the solution space.

For the balancer example one could for instance specify that the balls in all tracks should have the same diameter (Fi1 = Fi2 = Fi3) and that track three has fewer balls than track two, which has fewer balls than track one. The former constraint simplifies assembly and inventory, the latter improves the efficiency of the mass of the ball sets. These are common sense restrictions from a designers point of view and would reduce the solution space by orders of magnitude.

Constraint based systems is a common choice for configuration problems which by nature have discrete design variables (selections) and a finite solution space. Today a potential buyer can customise many products using the Internet. There are configurators intended for customer use for products ranging from bicycles and PC:s to motor cars and normally there is some sort of constraint solver working in the background to make this possible.

To summarise the characteristics of methods to store and process explicit knowledge the:

- **Procedural** processes require that a predetermined sequence of operations can be found (by the programmer) that leads to the desired solution
- **Inference** (or rule) based processes assumes that a sequence of operations that leads to the desired solution can be found (automatically) at run-time. Solution sequence is determined during execution. Knowledge structures with cycles can not be solved

- **Generate&Test** algorithms creates all possible design solutions first, then removes those solutions that do not satisfy given constraints. Discrete variables and finite solution domain is required. Combinatorial explosion will make the method unfeasible for all but very small problems!
- **Constraint processing** improves the efficiency of exhaustive search by pruning the search tree according to specified constraints. The remaining nodes of the search tree are used for G&T. Solves coupled, multidirectional problems with finite number of possible solutions. The design variables must be finite sets of discrete numbers or symbols.

References

Coppin, B.: Artificial Intelligence Illuminated. Jones and Bartlett Publishers, UK (2004)
Sunnersjo, S., Cederfeldt, M., Elgh, F., Rask, I.: A transparent design system for iterative product development. J. Comput. Inf. Sci. Eng. ASME **6**(3), 300–307

Chapter 7
Representation and Processing of Implicit Knowledge

Design tasks where all design rules can be derived and formulated stringently from basic physical principles are based on insight, understanding and control. This desirable situation implies that it is possible to compile a knowledge base that explicitly and completely describes the knowledge required to design a specific product. Such is the case for some engineering products, e.g. the automatic balancer previously used as example, and the representation and processing of explicit knowledge was described in the preceding Chapter. However, to really investigate and formulate design rules for all aspects of a product range might in practice either be out of reach technically or costly out of proportion. In the real world engineering design rely to a considerable extent on less rigorous knowledge. Insights in the form of experience, rules of thumb and common sense play an important role. One way of describing this is to see the knowledge as implicitly embedded in successful products, test results, user experiences or even in the design problem formulation. To produce a design proposal in such cases this knowledge needs to be acquired and applied in its implicit form. Three search based methods with ability to represent and process implicit knowledge will be discussed in this Chapter.

The distinction between explicit and implicit knowledge processing is clear cut in many cases but not in all. If we return to the six knowledge categories of Chap. 4 we can conclude that *tacit* knowledge and knowledge based on *comparison* clearly are of an *implicit* nature. Methods that use mathematical tools are stringent and rigorous in themselves but may in some cases represent implicit knowledge embedded in experimental data, as in *neural networks,* or optimal solutions found by systematic search through *hill-climbing* or *genetic optimisation*. It thus seems logical to include optimisation algorithms, neural networks and other interpolation algorithms in the present Chapter.

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_7

7.1 Case Based Reasoning, CBR, and Case Based Design, CBD

The method of Case Based Reasoning, CBR, is widely used in general AI applications to store and retrieve experiences and "lessons earned". The method evolved during the 1970s and 1980s and is generally attributed to Robert Schank, see Schank (1983). Many help desk support systems use some version of the CBR method which is then mainly based on storage of text strings, symbols and logic arranged in a tree structure. In engineering design one important field of application is to store, retrieve and reuse previous design solutions that have proved successful.

This focused application of CBR, which will be referred to as Case Based Design, CBD, thus tries to reuse existing solutions whenever possible rather than creating new designs—the general assumption being that *similar design problems have similar design solutions*. The term "reasoning" is somewhat exaggerated—their isn't much reasoning involved in the process. Instead, the underlying idea behind CBD is to abstain from using explicit product knowledge and base the design of new product variants on the knowledge embedded in similar, existing products. For a product that has been in use for a significant length of time and performed well it can be concluded that its design features, dimensions, material composition and so on satisfy customer and other requirements well. This conclusion can be drawn without having any detailed insight into the actual relations and constraints that govern the problem.

For a customer order of a new variant from an existing product family, it is a sensible thought to start by looking for previous, similar solutions and make minor adaptations rather than start from scratch each time. The origins of and relations between all design variables are often not fully known or understood in the sense that they are not derived from fundamental physical principles. The actual design knowledge is never explicitly stated when using the CBD approach, instead the existing, proven solution serve as a design guideline.

Obviously there are pros and cons with this strategy. On the one hand the proven reliability of an existing product together with low costs and short lead times for product development and the motivation to avoid management of unnecessary variants are significant advantages. On the other hand insight into product knowledge promotes improvements and innovations leading to a gradual evolution of the product. Normally companies working with CBD technology for this purpose will also do parallel work based on explicit knowledge. New product lines as well as cases not found in the case base will make certain that such traditional design work will never be superfluous.

7.1.1 Establishing a Case Base

In an engineering design office all company products will be documented by e.g. drawings, bills of materials and technical specifications. This documentation shows *how* the product should be designed and manufactured and is next to 100 % complete. Very little is however documented in a structured way about *why* a certain solution was chosen, why a certain surface treatment was selected or why specific dimensions were decided. This explicit knowledge is typically described in notebooks, reports or files in an unstructured way or in the memories of experienced designers. The information in this form is useless for the computer not only because it is unstructured, but also because it is probably incomplete. Instead, the use of CBD implies that existing solutions with a high degree of similarity to a required new variant are retrieved from a relational database, which is called a *case base.* Although the knowledge describing and defining the attributes of a specific case is implicit, the CBD method requires extensive, but not necessarily complete, information about the problem formulation, solution chosen and relevant circumstances.

Some kind of computer support is needed to search the case base since the stored solutions will normally be too numerous to be managed only by the memories of the designers involved. What is searched for is not necessarily a perfect match (which probably does not exist), but finding a small number of candidate cases which are close to what is searched for among a large number of existing designs.

The final selection among these candidates is then done manually by the designer, who also takes the decision whether the best match is close enough and can be used directly or decides if modifications might be necessary. In some cases no acceptable candidate is found and an entirely new variant must be created.

As cases are retrieved, modified, used and found to be successful, these updated cases are added to the case base, thereby extending and modernising the repository of available design options. Thus the case base represents a living storage of implicit knowledge and will reflect changes in market, technology and current engineering practice.

To facilitate the search process the design variants must be indexed in a purposeful way. Indexing could be symbolical or numerical or both. The important thing is that indices accurately represent the crucial properties and characteristics of the design so that the search and match process is driven only by the attributes that really matter. Figures 7.1 and 7.2 illustrate indexing of a roof rack bracket from case No 4 in Appendix A, and shows how the product range is divided into five topological variants that are distinct and can be clearly defined. Each of these topologies have a set of parameters that represent the main dimensions that are crucial for the functionality of the product. Selecting and defining the indices require in-depth understanding of the product and how it works and thus requires active involvement of the designers concerned.

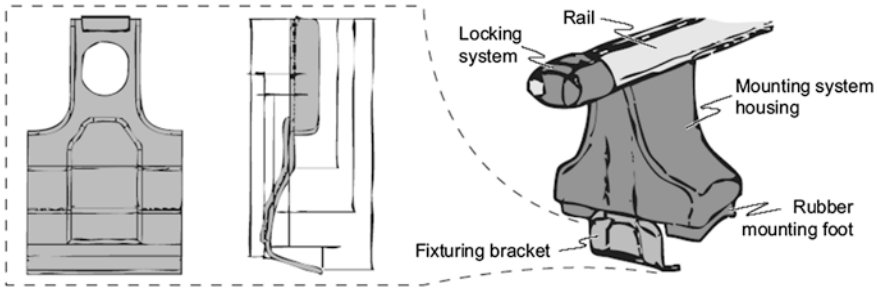


Fig. 7.1 Roof rack attachment with fixturing bracket. From Cederfeldt (2007)

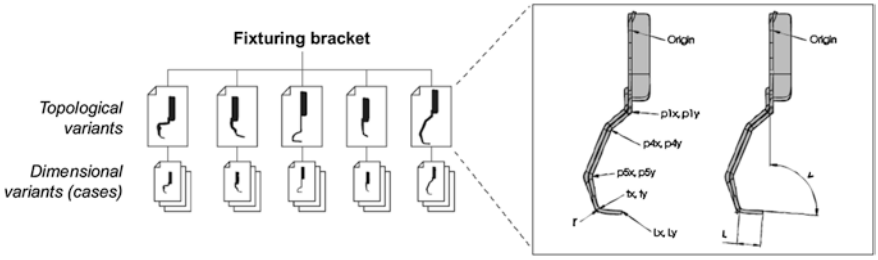


Fig. 7.2 Range of brackets divided in natural classes depending on topology or concept. The existing cases are sorted into these classes and ready to be searched for best match. From Cederfeldt (2007) in cooperation with Thule Sweden AB

The governing parameters are stored on a template, one for each case in the case base. These templates should also contain meta data like references to corresponding drawings, tools, control instructions, laboratory test reports and any certificates that might be required. Obviously this information must be strictly structured so that it can easily be captured and stored in a uniform way for each new case. There must be consensus among the users that on the one hand all relevant information is captured by the set of attributes selected and on the other hand, that the process is not made unnecessarily laborious by including attributes of minor importance. The challenges in this respect comes over time once the system is established. If in retrospect it turns out that additional attributes should have been included, this will be very difficult and time consuming to put right.

7.1.2 Case Retrieval by Search and Match

Cases are typically retrieved in two or more steps. The primary *search* process scans the case base for one or several class(es) or concept(s), that are relevant for the new variant. This search is normally based on symbolic parameters with the

binary values one or zero—either the class applies or it does not. The calculation of the similarity factor, SIM, thus becomes,

$$SIM_{symb;i}(a,b) = \left\{ \begin{array}{ll} 1; & if\ a = b_i \\ 0; & otherwise \end{array} \right\}$$

where a denotes the class of the new variant and b_i is the set of classes of the stored variants.

When a relevant class is found, the second step of the retrieval process will rank the cases of this class according to how similar they are to the new specification. This ranking process will use a *matching* calculation to determine the similarity factor, $SIM_{case\text{-}No}$. This metric is calculated as the weighted sum of the contributions of each attribute depending on how well it conforms to the goal value. This coincidence is calculated using a window that attenuates the contribution depending on how far from the goal value the evaluated case attribute is. A simple and popular formula for calculation of similarity between a parameter for the new product, P_{new} and a parameter for an existing product, P_{old} is,

$$SIM(P_{old}; P_{new}) = \frac{a}{a + \sqrt{\frac{(P_{new} - P_{old})^2}{P_{new}^2}}}$$

i.e. for identical parameter values the similarity is one, otherwise it is between one and zero. A large deviation from the goal values brings the similarity close to zero. The total similarity metric for case Q with k indices is calculated as the weighted sum of all contributions and then becomes,

$$SIM_{case\_Q} = \frac{\sum_{j=1}^{k} w_j \cdot SIM_{Q,j}}{\sum w_j}$$

where w are the k weighting factors representing the estimated importance of parameter j and summation is made over k parameter values.

Another simple window function was set up in application case No four in Appendix A. Here a modified root mean square calculation was used and this function is illustrated in Fig. 7.3.

Selecting window function and weighting factors is a matter of subjective estimates and trial and error must be relied on. It is important that the similarity values for the different cases really differentiate, but the result is not critically dependent on the choices of weight factors and constants, since the purpose only is to identify a small group of promising candidates. The final decision must always be taken by a knowledgeable designer who takes the final responsibility for the design.

$$x_{old} := 1..200 \qquad x_{problem} := 100 \qquad a := 0.1 \qquad sim_e(x_{old}) := \frac{a}{a + \left(\frac{x_{problem} - x_{old}}{x_{problem}}\right)^2}$$

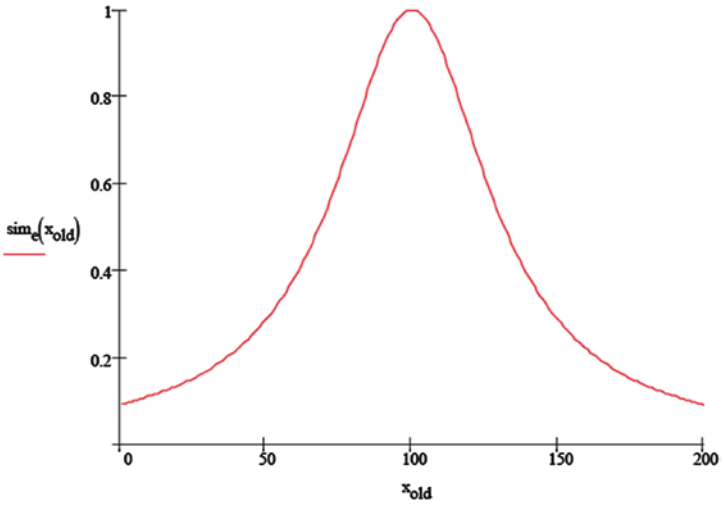


Fig. 7.3 Window-function for case matching used for application case no 4 of Appendix A

7.2 Interpolation Methods

One way of looking at a product is to see it as an input-output system (see Hubka and Eder in Sect. 2.2). This approach to modelling seems natural for products like pumps, engines, fans, heaters and so on. To model such a product includes the establishment of a transfer function that accurately simulates system response. There are engineering systems with many input parameters, many output variables and complicated relations between the two. Assume that we for the purpose of determining a value for some crucial product property need to create a digital model for such a system. In cases where these relationships are possible to derive analytically or determine by simulations, this would be the preferred strategy. There will however be many cases where such derivations are difficult to establish but where experimental data in the form of sets of inputs and corresponding sets of outputs exist. The rules and relations governing the effects of these systems are implicit and hidden in the experimental data and the use of such data offers an alternative to direct, explicit modelling. The model is based on discrete points from experiments (or simulation) but will return property values for a continuous range of parameter values—a process of interpolation.

7.2.1 Curve Fitting and Response Surfaces

For a system with one or two input parameters and one output variable, the relationship between input and output could be represented by curve fitting or response surface techniques provided the functions are reasonably smooth and well behaved. For a linear approximation of system properties a function f is defined as,

$$f(x, y) = a_0 + a_1 x + a_2 y + \varepsilon$$

where x and y are the input parameters, a_0, a_1, a_2 are constants to be determined and ε is an error function that will represent deviations between model and experimental data. The same approach can be generalised to systems with nonlinear properties using a second order surface definition,

$$f(x, y) = a_0 + a_1 x + a_2 x^2 + a_3 y + a_4 y^2 + a_5 xy + \varepsilon$$

The linear system requires at least N + 1 value pairs and the quadratic system at least (1 + N)N/2 + N + 1 value pairs. It is preferable to use more empirical data than this, about 1.5 times the minimum, to average out fluctuations of the data. The error term is written,

$$\varepsilon(a_1, \ldots a_N) = \sum_{i=1}^{N} \left[f_i^e - f(x, y) \right]^2$$

where f_i^e is the experimental value Number i. To determine the constants, a_i, the error term is minimised by differentiating the function with respect to the sought constants and then setting these derivatives to zero. When the system of equations is solved the constants giving minimum error term will result.

Using polynomials for curve and surface fitting is the standard approach. For those cases where physical insight suggest a better approximation there is every reason to use this instead. An example from the field of acoustics will illustrate this approach. Figure 7.4 shows how a pressure pulse in water dies out with distance. Say that we need to define a continuous pressure function, $P(r)$, that will reflect how pressure attenuates with distance, r. From acoustic theory it is known that also a complex sound source at a distance can be well approximated by contributions from a monopole (pulsating sphere) and a dipole (oscillating sphere) pressure sources. Monopole pressure is reduced in inverted proportion to the distance, while a dipole is reduced in inverted proportion to the squared distance. The assumed pressure pulse function is thus written

$$P(r) = \frac{a_1}{r} + \frac{a_2}{r^2} + \varepsilon$$

where P is a complex valued variable that represents magnitude and phase of the pulse. The function was fitted through seven measured values under the hull of a ship and resulted in the graph of Fig. 7.4.

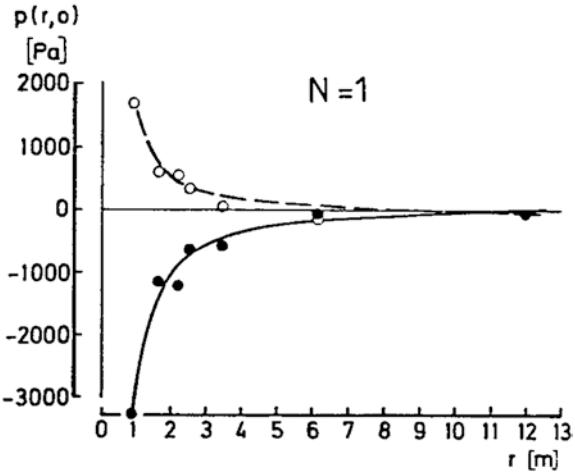


Fig. 7.4 Example of curve fitting: pressure pulse magnitude as a function of distance from source. Real part (*white*) and imaginary part (*black*). From Sunnersjö (1983)

Response surfaces based on polynomials higher than two and having many variables become impractical. Hence, when the relations have discontinuities and strong non-linear effects a more powerful modelling method is needed. One option is the use of neural networks.

7.2.2 Neural Networks

The purpose of the network is to create a set-up which will respond to inputs and generating outputs similar to the original system. Use of neural networks has been an active research field since the 1940s and the technology is applied to a wide variety of problems like speech recognition, robotics, pattern recognition and diagnostics. There is an abundance of literature on the subject. This section will give a brief overview of using the method in design automation. For guidance on how to build neural network systems in practice the reader is referred to e.g. Matlab Neural Network Toolbox.

The concept is inspired by the structure of the biological brain tissue. Such tissue consists of neurones being interconnected in an enormous network. Connections with incoming signals to the neurons are called dendrites and connections for outgoing signals are called axons. The actual functioning of the human brain is not known at the detailed level and it may well be that the similarity between biological brain tissue and computerised neural networks is superficial. Nevertheless, neural networks are composed of a set of artificial neurons that are interconnected in a way that reminds of the dendrites and axons of the cerebral cortex.

Consider the network, called a Perceptron, in Fig. 7.5. All incoming signals to respective neuron are multiplied with a weighting coefficient, w_i, and are then

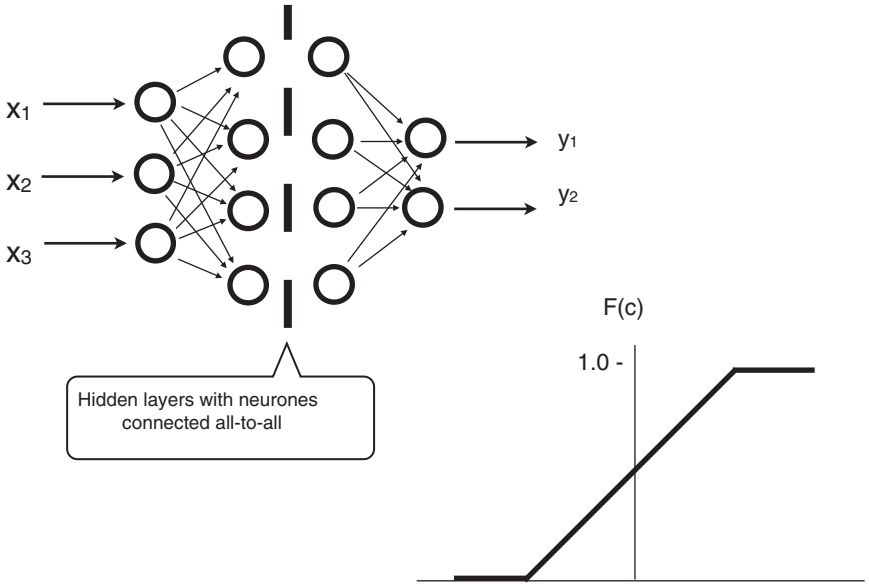


Fig. 7.5 Neural net with three input parameters, x_i, and two output variables, y_i (*top*) and ramp as neurone transfer function (*bottom*)

added to form the total input, C. For neurone Q with N inputs the cumulative input is given by,

$$C_Q = \sum_{i=1}^{N} w_i x_i + w_0$$

Each neuron has a transfer function which relates the C_Q value to the output value. Such transfer functions could e.g. have the form of a ramp, as in Fig. 7.5, or a step. It is however advantageous to use a transfer function that is differentiable to simplify the optimisation process that is done when "training" the system. Therefore the Sigmoid transfer function, which reminds of a smoothed ramp, is often used and is defined by

$$f_s(C) = \frac{1}{1 + e^{-C}}$$

Figure 7.5 shows one example of how the neurons are arranged in a network. There are a large number of variants of nets possible and the choice is not trivial. In practice one has to rely on a trial-and-error procedure. The net shown in Fig. 7.5 has two outer layers that account for input and output respectively. They have one neuron for each input parameter and correspondingly for the output variables. In between these layers are a number of hidden layers. All neurones in the hidden layers are connected to all neurones in the two adjacent layers. There are no connections between neurones in the same layer.

A neural net works in two steps, *training* and *prediction*. When training a net a number of given input/output sets known to be true are presented to the net. The weighting coefficients of the net are varied iteratively to achieve best average correspondence between net predictions and true values. This is an optimisation problem with the weight coefficients as design variables and the squared differences between predictions and true values as the objective function to be minimised. If the transfer functions are differentiable, the stationary points can be found by solving the system of equation with the derivatives of the objective functions set to zero. One would think that more training values would result in better modelling., but what actually happens is that the predicted response coincides better with the discrete, given data points, but might on the other hand exhibit a "jumpy" unphysical behaviour. This is known as "overtraining".

Once the weighting coefficients have been determined so that the net works with sufficient accuracy it can instead be used for predictions of response to new sets of input data to determine reliable predictions based on interpolation. As with all interpolation the results are reliable only within the parameter ranges used for training of the system.

As previously discussed the neural net is an alternative to analytical derivation or numerical simulation where the real physics of the problem is too complicated or time consuming to clarify. For such cases the net could form a part of a design automation system used to estimate required properties. Further use could be where simulations are possible but very time consuming. If a large number of design points are needed for such cases, e.g. for iterative optimisation, determination of robustness or stochastic properties all interpolation methods could be used to represent system response. Based on a small number of data points this digital system could return correct responses in a fraction of the time needed for each new simulation.

7.3 Optimisation

Optimisation is a wide field of heuristics and mathematics used to determine the "best" solutions for numerical problems within given constraints. Three of the more common numerical methods will be briefly introduced in this section. It is natural to categorise the knowledge used as implicit since the domain specific knowledge is embedded in the relations and constraints governing the design variables of the problem and the solution is determined by a search process.

Design tasks with conflicting requirements, e.g. simultaneous requirements of low weight and high strength, often forms the core of a design problem. To strike the best possible balance using analytical or heuristic methods can sometimes be difficult. Often an iterative search process is needed, which step by step converges towards an *optimal* solution. From Chap. 5 we remember that a box in the DSM often indicates that there is an underlying optimisation problem.

Optimisation is therefore a tool often required as a part of a design automation system, usually in combination with a parametric CAD model.

An optimisation problem is defined by the following characteristic factors, which need to be clarified before attempting a solution,

- **Optimisation objective**. For a single objective problem: What property or function should be optimised (e.g. weight, strength, cost, efficiency, performance)? For a multiple objective problem: Which combinations should be optimised (weight and strength)?
- **Design parameters** (**fixed**). What parameters are defined as input for the problem (e.g. dimensions, surrounding temperature, power)?
- **Design variables**. What design variables should be varied to obtain maximum of the objective function? Are these variables continous or discrete? Define **driving** (independant) and **driven** (dependent) **design variables.**
- **Fixed conditions and constants**. Force of gravity, natural laws, standards.
- **Relations and constraints**. Define the mathematical or other relations between governing parameters and variables and between variables themselves.

To solve this kind of problems there are heuristic, analytic, numeric and genetic methods. The methods used for optimisation is of course equally applicable to minimisation—it is only a matter of changing sign of the objective function. The review below is focused on numeric and genetic methods.

7.3.1 Hill Climbing Methods

The young man in Fig. 7.6 is going to try to find the highest point (optimum of the objective function) of the hill. Since he is blind-folded he must employ some sort of search strategy since random search would be far too inefficient. The principle of hill climbing as an optimisation method implies starting at a point within the solution space, finding the direction of steepest ascent and taking one step in that direction. The process is repeated until the top is found and then terminated. The position of the climber is defined by the x and y coordinates of a horizontal coordinate system (design variables). Since the topology of the hill is uneven, the direction needs to be redefined for each new step. The search space is limited by the fence (constraints) which must not be crossed. The stepwise climbing will eventually lead the climber to the vicinity of the goal where the slope right at the top is zero. This is the criteria for the optimum being found and the search should terminate.

The most straightforward method for hill climbing is called Complex and is a non-gradient method, see Andersson (2001). The process starts using a group of three random initial values for a problem with two design variables, see Fig. 7.6. The start values are inserted in the objective function and the three objective values are determined. The point with the lowest resulting objective value is removed and replaced by its reflexion through the centroid of a line drawn between the two

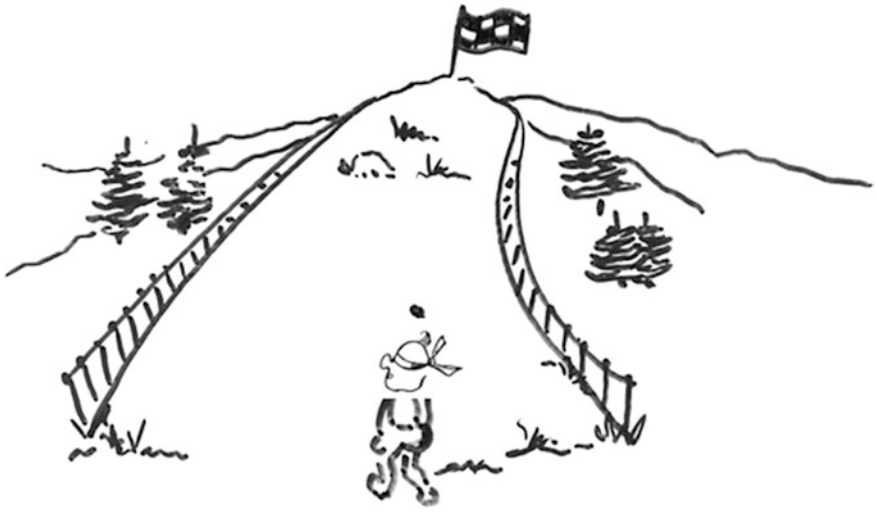


Fig. 7.6 Finding highest point by blind folded "hill climbing"

remaining points. This process is repeated over and over again and thereby making the group of points start climbing uphill and gradually find its way towards the maximum until convergence to optimum is achieved.

Complex is a robust but somewhat slow optimisation method. For cases when the objective function is differentiable a more efficient search method is to use the gradient of the objective function to determine the direction of steepest ascent. Both search methods start from the same problem definition:

Objective Function

$$G(x_1; x_2; \ldots x_N)$$

Constraints

$$C(x) \leq R_i$$

Solution space

$$x_i \in X$$

Start value

$$x_{0;i}$$

In addition, the gradient method also requires the objective function to be differentiated with respect to the design variables,

$$Grad(G) = \left[\frac{\partial G}{\partial x_1}; \frac{\partial G}{\partial x_2}; \ldots \frac{\partial G}{\partial x_n}\right]$$

The hill climbing proceeds in steps in the direction where the slope is steepest just like the Complex method. The step directions are determined for each new step by inserting the design variables of the current position in the expression for the gradient of the objective function. This method will normally converge faster than Complex. Both these methods, or variants of them, are widely used and they are easily available as parts of math kits like MatchCad or Matlab.

As Fig. 7.6 indicates the hill climbing process is blindfolded and the user needs to pay attention and not take the results for granted. First, it may well be that the highest point is at the solution space boundary. This optimum will not be characterised as a stationary point with slope equal to zero. In such cases the normal criteria for search termination will not be relevant and need to be complemented. Secondly, there is always the risk of climbing a local optima within the solution space leading to the search process terminating before the global optimum, i.e. the highest point in the entire solution space, has been found. There is no panacea for either of these problems except human monitoring and intervention.

7.3.2 Genetic Method

As the name implies this method uses the evolution process in nature as a model for simulation. Optimisation by genetic search has proved to be a reliable and efficient method which is not bothered by discontinuities of the objective function or getting trapped in local optima. In the same way as natural evolution however, the convergence can be slow and one cannot be certain that the search converges at all. Starting the optimisation at new coordinates will probably give different results, but usually not drastically different.

The initial, start solutions are seen as individuals in a population. The performance of each individual is determined by an objective function, or in the terminology of genetic optimisation, the fitness function. The individuals with poor performance are removed from the population and new individuals, (children) are created by mating pair of individuals (parents) that perform well. By systematically removing the least successful individuals and replacing them with off-spring from the best parents each new generation will gradually reach better and better values of the fitness function.

The design variables are organised in groups, each associated with a component or a group of components. The design variable values are written as binary strings and assembled to form a *gene* that defines the current properties of the component design. Together all genes form a chromosome that fully defines the product.

The evolution is based on the principle of survival of the fittest. Thus a fitness function will be used to determine which solutions that should be retained and which should be removed from the population of solutions.

The genetic optimisation process in its simplest form follows the steps below:

1. Determine an initial population of separate design solutions with design variables randomly distributed over the solution space
2. Evaluate the objective function for all individuals
3. Retain the solutions that rate in the better half of the population and delete the remaining solutions
4. Select pairs of solutions randomly and combine their genes (mating) until all remaining solutions have produced offspring. The genes of each of the parents are divided, e.g. in the middle, and swopped so that if parent 1 has a gene A;B, and parent 2 has a gene C;D, then the first child will get the gene A;D and the second child the gene C;B
5. The process returns to step 2 and a new iteration is started. The changes between iterations are monitored and when the results appear to not improve further, the process is terminated.

How a genetic optimisation run progresses is not so easy to foresee and the process can be improved by trial and error. To avoid getting stuck in local optima the genes are occasionally mutated, i.e. a small, random disturbance is added e.g. every hundred generation in order to continue search for the optimum. It is often beneficial not to mate all individual solutions and instead leave the best 10 % unchanged ("elitist search"). Despite a degree of uncertainty that is associated with the properties of the method, the practical experiences are very convincing. It is obvious that evolutionary processes have an inherent stability and effectiveness. For guidance of the practical use of the method, see e.g. Matlab Toolbox on Genetic optimisation.

References

Andersson, J.: Multiobjective optimization in engineering design—applications to fluid power systems. Dissertation No. 675, Linköping University, Sweden. 2001LiTH-IKP-R-1097 (2001)

Cederfeldt, M.: Planning design automation. PhD thesis, Chalmers University of Technology, Göteborg, Sweden (2007)

MatLab Neural Network Toolbox, http://www.mathworks.com/products/neural-network/

MatLab Genetic Optimisation Toolbox, http://www.mathworks.com/discovery/genetic-algorithm.html

Schank, R.: Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge University Press, USA. ISBN:0521248582 (1983)

Sunnersjö, S.: Propeller induced hull vibrations—the determination of exciting forces, Proc Inter-Noise 83, Edinburgh (1983)

Chapter 8
Planning a Design Automation System

When product variety is a driving business factor success relies on skilful creation of:

- A family of products that are modularised and/or parametric,
- A production system with a high degree of flexibility
- Standardised design procedures implemented in supporting computer systems

The focus of this text is the third point. Aspects relating to product design were briefly discussed in Chaps. 2 and 3, while the production aspects and their interaction with the product design is entirely left out here—it needs a book of its own. Application cases with systems implemented in industry and academia are presented in Appendix A.

This chapter is intended to demonstrate how to apply the tools and methods previously presented in this text to set up a well functioning, reliable, maintainable and cost effective design automation system. Developing such a system is a considerable investment in time and money and once implemented it will be a crucial factor in the business of the company. Figure 8.1 illustrates a process which narrows down the original wide field of options to a system tailored for the needs of a specific company.

The basic principles of knowledge processing in engineering as previously described are implemented in most commercial software. Companies with sufficient resources and needing something not readily available might choose to write their own programming language, which may be based on these principals or some variation. Such software will be dedicated to a specific problem domain, e.g. creation of a certain type of 3D geometry in a given CAD system. Building a library for frequently occurring functions greatly simplifies setting up DA systems for new products and product variants and reuses the same code to save time and money. These problem specific languages, which are company proprietary, are generally called Domain Specific Languages, DSL, and are used for many types of applications also outside DA, see e.g. (Fowler 2011).

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4_8

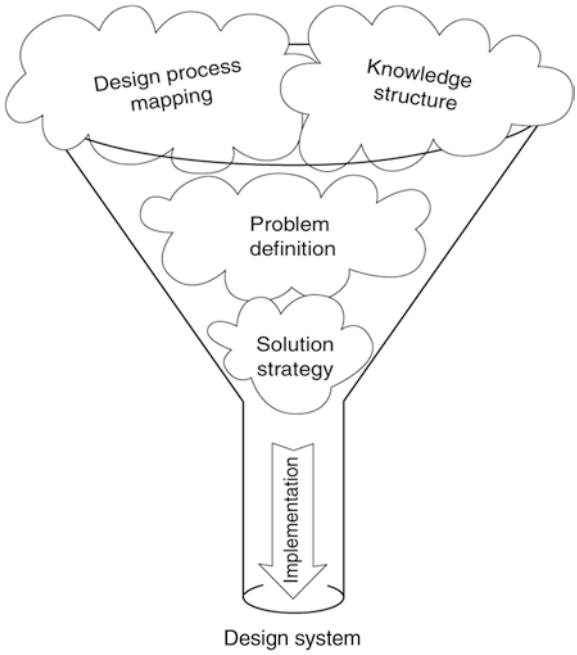


Fig. 8.1 A top-down process to create an automated design system

8.1 Seven Steps for Systematic Planning of a Design Automation Project

For an energetic management it might appear tempting to acquire a commercial system off the shelf and get started right away with the coding. Often the CAD or PDM systems that the company already uses have plug-ins for some kind of design automation and selecting such a system might appear to be the obvious solution. There are however good reasons why some time should be spent to get a reasonable understanding of what the design tasks require in terms of knowledge representation and processing and how the intended system should work before thinking in terms of commercial software.

Experience shows that many DA systems have had to be rebuilt from scratch after commissioning or only being used a short time because the need of important functions might not have been anticipated from the beginning or are not working properly. Also, building a first small system, maybe a prototype, is a learning experience that will clarify many aspects of the intended project and will help to avoid the real stumbling-blocks.

A project to set up and operate a DA system is usually led by a *knowledge engineer*, who could be an experienced engineering designer with an interest in computing or a programmer with an interest in design and product development. Below are listed seven items worth considering before taking any definitive decisions on the project, let alone selecting specific software.

8.1.1 Get Acceptance and Involvement

The establishment of the relevant domain knowledge is entirely dependent on the cooperation of experienced designers—the knowledge engineer is unlikely to possess sufficiently detailed knowledge. Unless the designers feel that the system will be to their advantage and get actively involved in its creation, the project will not be successful. There might be an apprehension to share design knowledge unless the system is seen as an aid in the designer's own work. It is thus important that the designers understand that they are the intended users of the system and that their views will be decisive for how the system is specified, developed and used.

Experience from industrial applications indicate that DA systems will abolish much routine work and allow the designers to devote more time on improvements of existing products as well as the creation of new products. It even indicates a somewhat different professional role, where the designers leave the direct product modelling work in front of the CAD screen and instead work more indirectly by creating design rules that the DA system will use to create the actual product definition. All in all, a DA system will benefit the business of the company but also make the individual designer's work more interesting and satisfying.

8.1.2 Define and Delimit Problem

One experience from many projects is that the tasks addressed are often too large—the total problem needs to be broken down into parts that can be solved and implemented one by one. If each subtask can be used stand-alone a modular structure will result naturally and the benefits as well as experiences of the system are obtained early. It is essential that the chosen tasks are not mutually dependent since this will result in cycles over the boundaries of the subtasks, which will need operator intervention to be resolved.

Early in the process a review should be made of how well the underlying technology is known and if a well defined work flow exists or is possible to develop. The final answer in this respect will not be known until late in the project but a knowledgeable opinion must be obtained before the project is defined.

8.1.3 Evaluate Cost/Benefit

This is a notoriously difficult part and serious investment pay-off estimates are seldom made. This is however something that could be said also about other complex IT investments. Except for straightforward configurators for catalogue design, the direct investment costs are difficult to estimate as is the long term costs and effects. The benefits are often easier to estimate at least in terms of reduced design

costs and lead times. To a considerable extent starting a DA project will therefore be based on the subjective convictions of a knowledgeable staff. If many of the following criteria apply, there are good chances that a DA system will prove to be a profitable investment:

- The products are sold in many variations
- Fast and correct quotations important
- Multidisciplinary design tasks cause delays because of difficulties to assemble specialists
- Reduced design costs important
- Reduced lead times important
- Poor development capacity because specialists are engaged in routine work
- Availability of domain specialists is a bottle-neck
- Quality problems because different individuals solve similar tasks by different methods
- System to manage corporate knowledge is valuable
- Design process traceability important.

8.1.4 Acquire Design Knowledge

This is the responsibility of the knowledge engineer, who will have to rely on the support of the experienced designers. It is one of the most crucial points in a DA project, but also an opportunity to review and improve existing design methods. Section 4.3 describes how raw knowledge is idealised and structured to provide a definition of domain knowledge to be used in coding. Sources for information will be text books and other publications, company specific design guide lines and, most importantly, interviews with the designers. Raw knowledge is acquired from individual designers or groups of designers while it is the task of the knowledge engineer to structure, formalise and document the knowledge and define when it is applicable. The programmer, finally, implements the knowledge in computer code for representation and processing.

The first step of this task is to agree on and define a common terminology, thereby assuring that the people involved in the project have the same understanding of the terms used. A simple thing like defining variable names that helps to intuitively identify the variable will in the long run prove important. The facts, rules and methods used to design a certain product must then be discussed, agreed upon and defined. Doubtless, there will appear examples where different designers have different views of how something should be done and this is the opportunity to agree on what should be the preferred working methods of the company. For documentation and security reasons it is wise to distinguish between generic and company specific domain knowledge.

This task can be expected to be a sizeable work that will require much time and many meetings with specialists. The outcome should be seen as a statement summarising the company's corporate knowledge of the product in question. As such it should be documented and authourised by the manager responsible for engineering, and it should be made available for all designers concerned. The methods for knowledge acquisition will be discussed in more detail in Sect. 8.3.

8.1.5 Clarify and Map Design Process

To clarify the intended work flow of the DA system the existing, manual design process is studied. The standard method for process modelling is the IDEF0 function modelling method, see e.g. Department of Defense: Systems Engineering Fundamentals. Which has its primary applications in production engineering. The method provides a formal representation of data flows using block diagrams with blocks relating inputs to outputs, see Figs. 8.2 and 8.3 which although not following IDEF10 standard, give an illustration of how these flow-charts are set up.

An alternative method for mapping of problem structure and information flow is the Dependency Structure Matrix, DSM, presented in Sect. 5.2.3. This method gives a good overview and also reveals characteristics that are significant for choice of DA method. How this method is used in practice will be discussed in Sect. 8.4.2.

8.1.6 Classify Problem and Select Solution Strategy

Having defined the knowledge to be used in the design process and how this process is structured, a suitable method for representation and processing can now be selected. Sometimes the problem requires a combination of two or more solution methods. Starting from an understanding of what knowledge categories that are required for the problem, the matrix of Sect. 4.5.2 can be used to identify suitable computational methods. To assure that the chosen method(s) will be able to process knowledge with the existing dependency structure, the discussion of Sects. 5.3 and 8.4.2 will provide guide-lines.

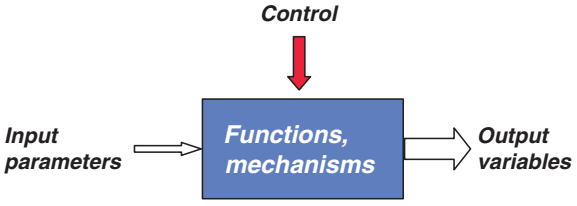


Fig. 8.2 Computable design task seen as input-output process

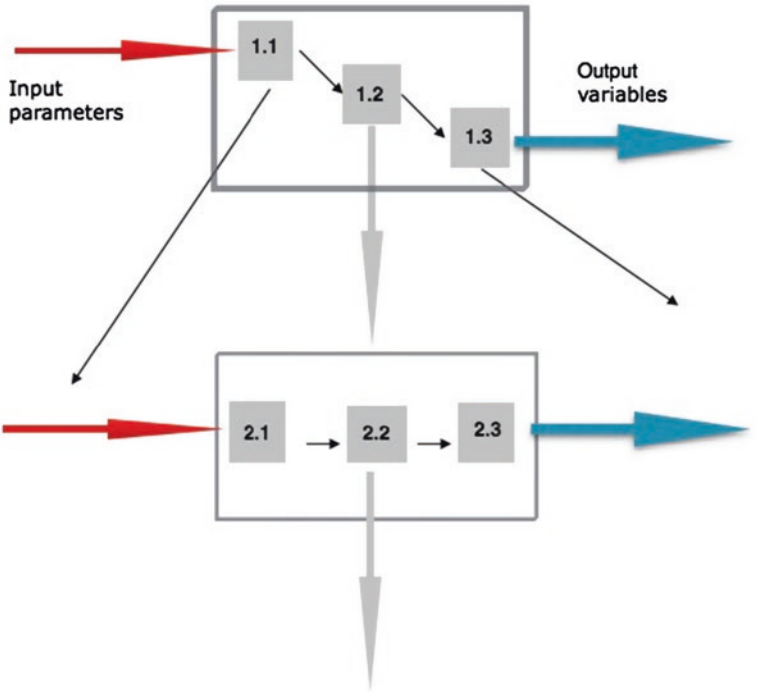


Fig. 8.3 Map of information flow. Details are represented by expanding the flow chart into required No of levels. The three subtasks on top are denoted 1.1, 1.2, 1.3. Subtask 1.2 is broken down into subtasks 2.1, 2.2, 2.3 and so on

8.1.7 Select Software Tools and Plan for Implementation, Maintenance and Expansion

Design automation systems can be programmed from scratch in any of the common programming languages, it could be bought off the shelf with a wide range of functionality already implemented or it could be something in between these extremes.

Building the system in-house is a significant undertaking, but has the advantage of giving full insight and control. Relying on commercial software always creates a risk of vendor dependence which may or may not be acceptable. There is a risk that software tools will be removed from the suppliers range of products and is no longer supported. If much resources have been used to set up the system in this software this is a significant set-back. It is a calculated risk and if a commercial software is selected there must also be a plan for what to do if it can no longer be used.

In most cases the DA system must be integrated with other design tools like CAD or PDM systems. Some of the suppliers of such systems also offer plug-ins for various functions of design automation. CAD systems focus on parametric

models which are driven by DA algorithms while PDM systems primarily support configuration design. There are of course also combinations available.

One possible compromise with regard to vendor dependence is to build the system from widespread and commonly used software units that are combined into a system with well defined interfaces. If one block expires it is a limited task to replace it with something new.

Any software, commercial or programmed in-house, might require replacement. Often however, the actual coding is not the dominating task in a DA project. Instead clarifying and preparing the knowledge normally is what consumes most resources. One way of reducing the effort when change of software is necessary is to have the domain knowledge carefully idealised and documented. If this is well taken care of a change of software is a limited effort.

Also when the same software can be used over many years there will be requirements of maintaining and updating the system. These program revisions must be done in a planned way at regular intervals except for acute errors that must be corrected immediately. Updating might be due to a need to,

- Correct errors that manifest themselves after commissioning of the system
- Improve user-friendliness of DA system
- Introduce new product ranges
- Introduce new product requirements
- Adapt to new product technology
- Adapt to improved computer technology.

8.2 Criteria for Evaluation

Having arrived at a proposal for a DA system the suitability can be evaluated according to the following five criteria, which have a general importance for all IT-systems,

- **Flexibility**—Ease of adaptation to new requirements.
  - Possibility to expand the system with more knowledge, more options and a higher degree of detail (scalability),
  - Possibility to introduce new variants, new geometry, new product structure or new solution methods.
  - Possibility to let the system grow to handle models that are much more complex than what will be tested in prototyping,
- **Longevity**—Realistic length of operational life.
  - Future software maintenance made easy with clear architecture, good documentation, functional transparency and widely used programming languages.
  - Vendor dependence for specialised software imposes risks that vital parts of the system might not be supported in the long run. What is the solution if this should happen?

- **User friendliness**—Ease of learning
  - Is the system easy to use for new users?
  - Do the interfaces with other software like CAD or FEM appear seamless?
- **Transparency**—Functions clear and understandable
  - Easy to access, read and understand the knowledge base used
  - A way of operation that the user can follow and understand
  - A program architecture that is easy to understand for new system developers that need to modify or update the system
- **Costs**:—Efficiency in terms of time and money
  - Initial investment in software and hardware
  - Costs for programming and testing
  - Costs for maintenance.

8.3 Knowledge Acquisition and Process Mapping

The stored design rules that govern the new product variants have been frequently referred to in this text. The composition of these rules will of course vary depending on application, but to give the reader an indication of what to expect in terms of rule actions a list originating from an investigation by Ryhanen (2004) of a DA system intended for a typical mechanical engineering product is quoted below:

- Rules that choose a configuration based on some condition.
- Rules that automatically create the part's geometry from the input specifications.
- Rules that calculate engineering properties about the part.
- Rules that optimise cost, performance, and quality.
- Rules that extract information from external databases.
- Rules that communicate with and analyse the results of external engineering analysis programs.

8.3.1 Compiling Design Knowledge by Reverse Engineering

For a company where products are normally planned in families and where variants are intended to be created automatically, the development of the DA system should proceed in parallel with the development of the actual product. When the need for automatically created variants arises for products that are already on the market the knowledge engineer will have a somewhat more difficult task to get access to the intentions and reasons of current design solutions.

In both cases however, the process could be compared to a process of reverse engineering, where the knowledge engineer is faced with the task of fully defining

a product by its design variables and derive their origin back to requirement specifications, laws of nature, algorithms, standards, guide-lines and personal opinions of experienced designers. The process will comprise the following steps,

- The starting point is an existing prototype or product
- Set up a Product Variant Master, see Sect. 3.3.1
- Define all design parameters, design variables, topological variations and configuration variations, see Fig. 8.4. Choose expressive parameter and variable names to make the knowledge base easier to understand
- Trace origin of all product defining information and their relations
- Clarify all relations between requirement specifications (customer, legal, physical, standards, manufacture…) and product definition
- Express relations in a form that provides a good knowledge representation for the intended computational method (procedural, rule based, constraint based)
- Document all results from the above steps.

The generic data flow of Fig. 8.4 is exemplified by an application for design of a cast housing for part of an industrial robot, see Fig. 8.5. The dimensions of the housing is determined by its internal components like gear wheels, bearings, shafts and couplings, which are chosen to satisfy the load carrying requirements. The 15 input parameters reflect the requirements and constraints and these are fed to an external inference based system which determines the required component dimensions—in all 171 variables out of which 44 variables define the housing. These in turn are by applying geometrical design rules, which are stored as part of the

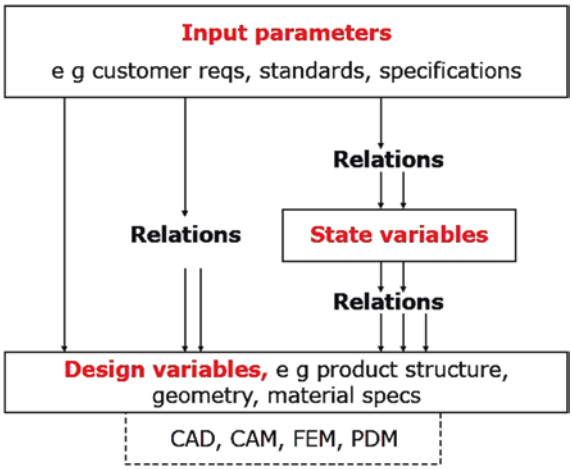


Fig. 8.4 Information flow in DA system where input parameters will multiply and generate the product definition (*design variables*). Input parameters might directly become variables (*left*), or by actions of relations, result in new variables (*middle*) or by actions of relations and intermediate (*state*) variables result in new variables (*right*). The design variables are subsequently transferred to other engineering tools

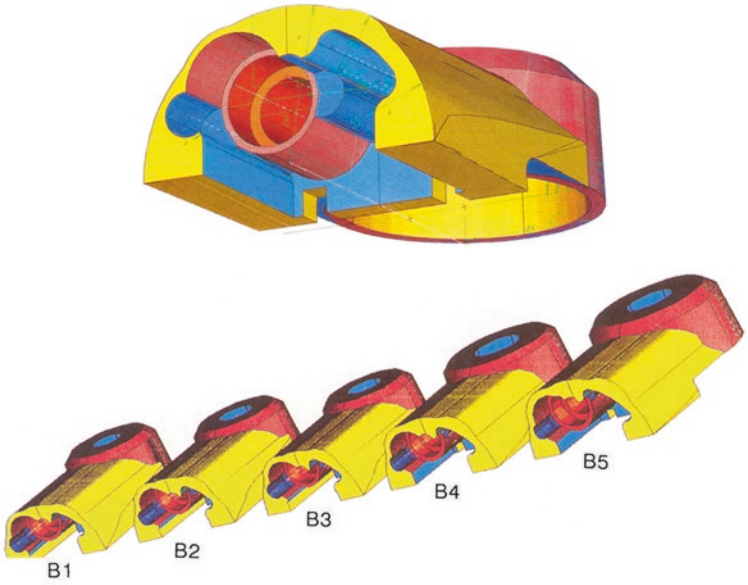


Fig. 8.5 Cast housing till robotic wrist. Basic design (*top*) and size range for different load capacity (capacity of B5 is twice that of B1)

CAD model, used to create the final geometry. This is defined by a total of 1334 driven design variables. The division of design rules between an external inference system and geometrical rules stored in the CAD system is a natural solution for problems with complicated geometry (Fig. 8.6).

8.3.2 Mapping of Design Process

In some sense the DA system will simulate the work of the designer(s). To accomplish this, the traditional, manual design process must be clarified and mapped. This is the natural starting point, but it is not self-evident that the DA system should copy the manual process—there might be different and better ways that suit the computer's way of working.

As mentioned in Sect. 8.1.4 there is a choice to carry out this process mapping either using IDEF0 method or to use the DSM method. IDEF0 gives a complete and standardized representation of the information flow and is supported by several software tools. One problem however, is that for complicated processes the diagrams become excessively complex and difficult to overview. Also parallel work and feedback loops are not easily represented. Although mapping using IDEF0 is feasible and correct, the following discussion will therefore assume that the DSM is used.

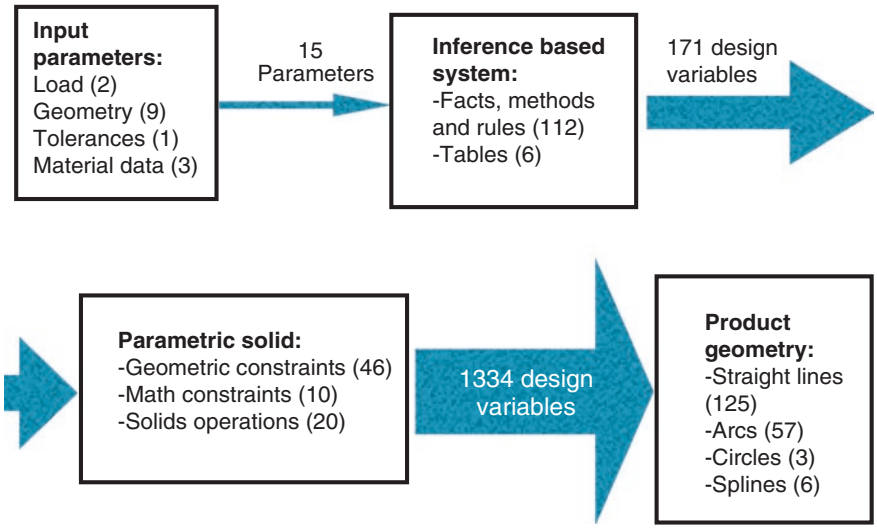


Fig. 8.6 Data flow for design of variants of the housing in Fig. 8.5. The input parameters generate intermediate design variables by applying design rules in a separate KBE system. The resulting variables are used to instantiate the generic CAD model

Graph theory lies behind many considerations that govern choice of solution method in DA. Graphs are however unpractical except for very small problems while DSMs, which carry exactly the same information, is a better tool to map and analyse real design processes. The DSM was originally a mathematical concept based on the adjacency matrix, which in turn is a compact representation of graphs with nodes and links. Today the DSM has become a tool of much wider use to analyse dependencies, see (Eppinger and Browning 2012). The DSM can be set up for different levels of detail and provides compact representation also of complex data flows.

DSM as a general method is used for two purposes: Clustering and sequencing. *Clustering* applies to e.g. modularisation of products and also to organisational issues. Both applications are outside the scope of this text. *Sequencing* or *partitioning,* on the other hand i highly relevant and reminds of the process of "topological sorting" that was discussed in relation to graphs in Chap. 5. Sequencing can be performed at several levels of detail, where higher levels are called *activity* based and the lowest, most detailed level is called *parameter* based. Activity based DSMs are necessary to provide overall view and to keep matrix sizes down. However, it is important to realise that if DSMs are to be used as process maps for coding, it is essential to reach also the parameter level. This is so for two reasons: Firstly, estimating dependencies at task level is prone to mistakes and secondly, many coupled tasks tend to be resolved, or at least be much simplified when brought down to parameter level, as is illustrated in Figs. 8.7 and 8.8.

	Items	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	Input																						
2	Tilt house;flangecoupling design	X		X																			
3	Tilt house;bearing 1 selection		X																				
4	Tilt house;wheel 6 design	X																					
5	Tilt house;wheel 2 design			X																			
6	Tilt house;shaft 6 design		X	X	X	X		X									X						
7	Main house;house design	X			X		X				X	X			X	X	X	X					
8	Main house;shaft 5 design		X					X															
9	Main house;bearing 5 selection														X								
10	Main house;wheel 3 design												X		X								
11	Main house;bearing 3 selection														X								
12	Main house;wheel 4 design	X						X						X			X						
13	Main house;relbearing selection	X							X														
14	Main house;wheel 5 design	X							X	X													
15	Main house;radbearing selection		X					X	X														
16	Main house;axbearing selection								X														
17	Arm;pinion design	X						X											X				
18	Arm;pinionbearing selection																	X					
19	Main house;carrier design								X				X		X	X	X						
20	Arm;s_bearing selection																	X					
21	Arm;house design	X						X										X					X
22	Arm;shaft design							X										X	X		X	X	

Fig. 8.7 DSM at task level for cast housing of industrial robot

Fig. 8.8 Coupled block above is expanded to parameter level and partitioned for best sequence. Note that the coupled block is resolved through partitioning so that a hierarchic data flow is obtained

	Items	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Input																	
2	Output from previous activities																	
3	SH;tjock (param, determ)	X																
4	SA;hojd1 (var, cont)		X															
5	SA;rad1 (var, cont)		X															
6	SA;rad2 (var, cont)		X															
7	SH;yr (var, cont)		X			X												
8	SA;hojd2 (var, cont)		X		X													
9	SA;ckant2 (var, cont)		X			X												
10	SA;bkant1 (var, cont)		X			X												
11	SA;bkant2 (var, cont)		X			X												
12	SA;rad (var, cont)			X		X												
13	SH;oms (var, cont)			X													X	
14	SH;ih (var, cont)		X	X			X											
15	SA;ydjup (var, cont)		X		X			X										
16	SH;idjup (var, cont)		X														X	
17	SA;ckant1 (var, cont)																X	

The characteristics of the DSM give significant indications of what processing methods that are suitable for a certain problem, see Sect. 8.4.2.

8.4 Planning Architecture and Working Principles

A complete design system including automatic functions will normally consist of several software components to handle geometry, manage product structure, carry out technical calculations and cost estimates, produce drawings, control tables and so forth. The mechanisms to represent and process knowledge is of course an

essential part of this system and how this is integrated with the other engineering resources require careful consideration.

There are commercial systems that claim to do all tasks in engineering design in an integrated environment. This sounds like the ideal solution, but nevertheless there are companies that prefer to build their own systems in standard languages using standard components. There are many aspects to take into account here—Ease of programming, interfacing with other programs, longevity, transparency, support, speed, cost and so forth. Some of these aspects have been touched upon earlier in the text and some will be discussed in this section.

8.4.1 Review of Options

It is certainly possible to build a system with dynamic rule processing using an ordinary programming language. However, when there is suitable commercial software available, using such software probably will save both time and money.

One of the largest and most extensively used systems described in this text, (application case No one in Appendix A) was written from scratch in Fortran IV in the 1970s including all graphics for drawings. Coding in a standard language like this is very time demanding but provides control and security. On the other hand, the flexibility for such a system is poor and maintenance is likely to become difficult over time.

Historically, the first generation of parametrical drawing programs came with the advent of 2D CAD systems with templates for input and macro languages to program geometrical and other relations. These simplified the programming of the graphics functions. The drawback was non-portable macro programs, often not even compatible between releases, requiring significant maintenance work.

With the arrival of 3D solid modellers with parametric functionality a further step towards automation was taken. Combined with a PDM system parametric and configuration design problems can be handled. These systems could be either procedural or inference based.

From the AI world generic rule based (expert system shells) and constraint based systems have been adapted for engineering purposes and either been given graphics functions or being integrated in a CAD or PDM systems. There are also dedicated design systems, e.g. for prefab houses or shipbuilding with many preprogrammed functions and also classification codes. Figure 8.9 illustrates the information flow where design rules are applied to input parameters in order to create a product definition.

A condition of significance for the system architecture is whether information flow between the modules of the system should be one-directional or bi-directional (see Fig. 8.9). In particular this concerns how the knowledge processor is linked to the CAD system. Consider as an example of a one-directional system the case where the parameter values from a spread-sheet or a specific template are transferred as predefined dimensions to a CAD model which is then updated. If e.g. the CAD model is used to calculate the weight of the product for selection of suitable

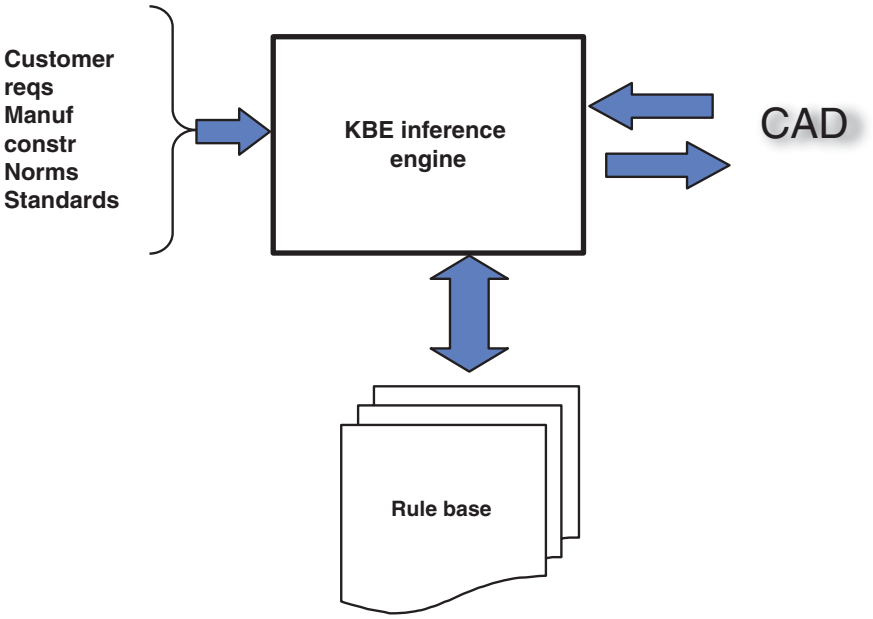


Fig. 8.9 Bidirectional data flow in rule based system in combination with a CAD system

fasteners, this information is fed back from the CAD system to the knowledge processor where the fastener selection is done. This would require bi-directional information flow. If the weight calculation is part of an optimisation loop this would require a close, bidirectional coupling. Also when a CAD model is built topologically under program control seamless coupling between CAD system and knowledge processor is needed. Such closely connected systems would need to use a system specific Application Protocol Interface, API. This discussion leads to the question that introduces the next section.

8.4.2 Where Should the Knowledge Be Stored and Processed?

Storing and processing knowledge is the central activity in a DA system. In what program module this activity should be carried out is worth some consideration. The question could be seen as a choice between two extremes, where the design rules are stored as:

- *Part of the product model*—the model "knows" how to design itself
- *Part of the design process*—the system simulates the design process with design knowledge available in the work flow.

Both these paradigms and combinations in between are widely used both for parametric design using a CAD system or configuration design using a PDM system. The choice is whether knowledge should be stored and processed inside the CAD/PDM system or in a separate program outside.

Speaking in favour of full integration is:

- Good computational efficiency of the system,
- having the same vendor for both CAD/PDM and knowledge processor clarifies the question of responsibility
- CAD system can be directly used in knowledge processing without any interfaces.

Speaking against it is:

- Vendor dependence for critical functions,
- poor transparency of system,
- limited to what is available in the existing tool kit and
- slow execution of macros.

Generally speaking the integrated architecture is mainly for products with few and simple components.

The alternative approach using external knowledge base and processor has advantages in terms of:

- Less vendor dependence,
- possibility to organise knowledge for better overview,
- full insight and control.

On the other hand, for simple products the system becomes unnecessarily complex and yet another system has to be maintained. For complex products and for systems where much of the computing is done by external programs this seems to be the natural method.

The two alternatives are illustrated in Figs. 8.10 and 8.11. The choice must be taken case by case, but it often appears natural to store geometrical constraints (parallel to, tangent to…) directly in the CAD model and external programs for non-geometrical knowledge and to govern the work flow. The practical solution will normally be a combination of the two principles, which however often means that there is a difficulty in that the design knowledge is divided into two systems, programmed by different persons. Coordination and maintenance require an extra effort.

One example of how design rules are divided between internal and external processing was showed in Figs. 8.5 and 8.6. The application requires design calculations and selections as well as geometrical rules and the knowledge processing is naturally divided into two groups. Figure 8.6 illustrates the growth of design variables as design rules are applied, which is the direct result of the automation effect of the DA system.

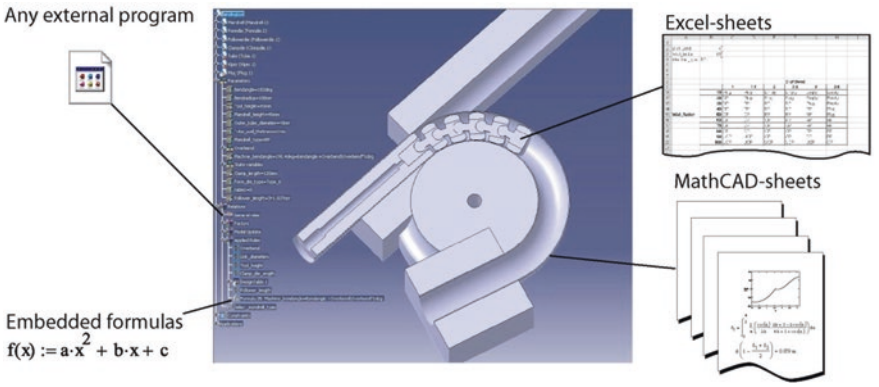


Fig. 8.10 DA-system for tube bending tools with design knowledge stored as a part of the CAD model. Processing by inference engine in the CAD system. From Johansson (2011)

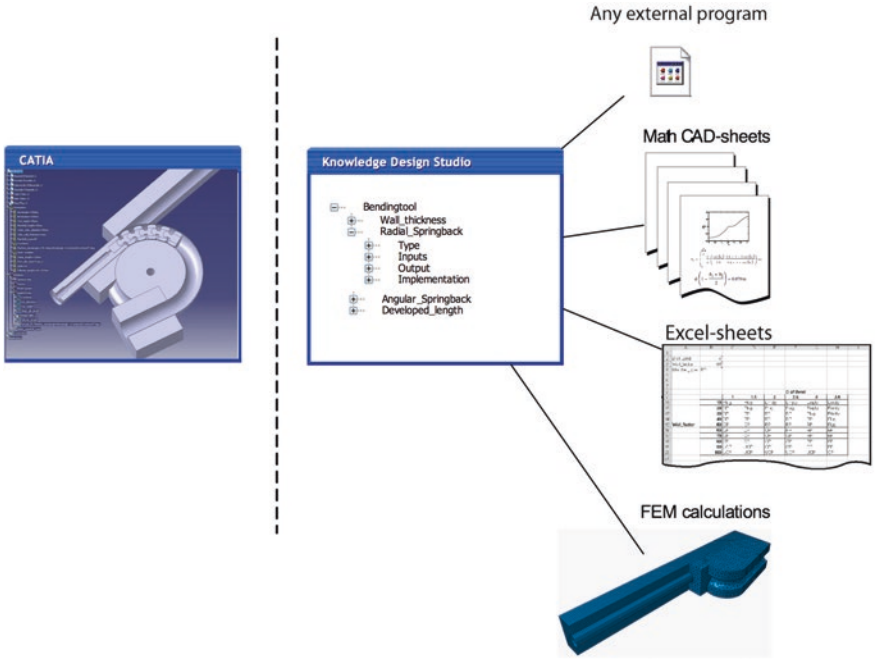


Fig. 8.11 As above, but knowledge processing in external, inference based system linked to the CAD using API. From Johansson (2011)

8.4.3 Matching Design Process to DA Methods

Consider the design task of Fig. 8.12. How would a designer solve this problem? There are essentially four different approaches:

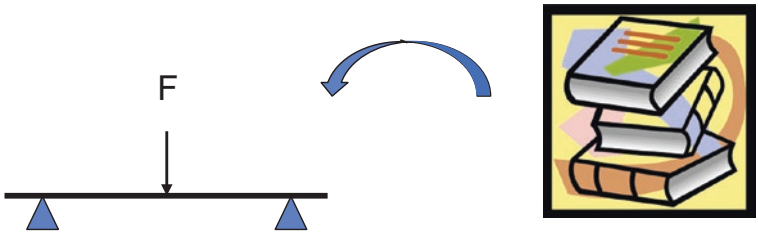


Fig. 8.12 Determine height of a beam with a quadratic cross-section so that deflection under force *F* is less than × millimetres (*left*). All necessary data and beam bending formulas are available i design manuals (*right*)

1. The designer has done similar tasks before and knows which formulas to apply and where in the handbooks to find them,
2. The designer finds a corresponding elementary case, starting from there he/she back-tracks and searches for all formulas required to insert into the elementary case and then calculates minimum required height,
3. As above, but the designer tries beam standard sizes until the smallest beam with sufficient strength is found,
4. The designer searches the files of finished projects and tries to find a similar case that is on the "safe side".

These four methods remind of four common DA approaches: Number one corresponds to the traditional procedural approach, Number two works like a rule based system with a backward chaining inference engine, Number three reminds of a G&T solver and Number four corresponds to the CBR, or rather, CBD, method.

When selecting which method to use for a certain design problem there are two fundamental issues to consider:

- How should the relevant design knowledge be represented?
- How should this knowledge be processed?

The first issue was discussed in Chap. 4 with a diagram in Sect. 4.5.2 giving cross references between knowledge categories and methods for computer representation. The second issue was discussed in Chap. 5, where it is shown how the characteristics of the problem structure in graph form could be used to select possible methods. While these conclusions were cast in a mathematical form, the practical recommendations from these studies are summarised below:

- Traditional computational problems with a process workflow that is predefined and fixed, with few conditional operations and when relations are sequential are best solved using straightforward *procedural* programming with tools like C or VB or algorithmic programs like MathCad, Matlab and so forth. This situation is often found for mature products with an exhaustive and a well researched foundation of engineering rules and is efficient also when heavy numerical processing is required.

- For problems with conditionally sequential relations (no cycles in graph or coupled boxes in the DSM) and/or where it is difficult to predetermine the workflow (e.g. complicated and nested conditional statements) there are three options:
  - The DSM is *partitioned* and resequenced so that the problem can be solved procedurally,
  - The relations are resolved at runtime using an *inference based* system.
  - If parameters and variables are discreet numbers with a finite solution space, a *constraint based* method is an efficient alternative.[1]

  The choice is dependent on whether the processing sequence can be expected to change at runtime and the expected need for such system maintenance that will cause changes of sequence. If this is the case it is usually advantageous to use rule or constraint based systems, since changes in the order of execution is much easier carried out. Further, the declarative knowledge base will be easier to control and manage as it allows free storage of design rules rather than in the order of execution.
- For cases with coupled relations (cycles in graph, boxes in DSMs, multidirectional processing) there are several possible combinations:
  - For very small problems with explicit knowledge and where parameters and variables are discrete numbers with a finite solution space, G&T is a possible option
  - As above but full size problems, the *Constraint based* systems which can resolve also problems with cycles, is an option.
  - When the variables are real numbers and when the best possible solution is required, the coupled part of the DSM should be solved through an *optimization* procedure.
  - When the relations cannot be expressed explicitly, but is based on the experience that certain mutually dependent variables form a set that define the product, then the natural solution strategy is *case-based reasoning*.

8.5 Documentation

8.5.1 Why Documentation Is Important

Documentation tends to be a neglected task that no one embraces with enthusiasm—designers shun bureaucratic work and management tend to see documentation as a non-value adding activity. Nevertheless it is absolutely necessary that the knowledge used by the system is fully documented and that this documentation is kept up to date when changes are made. There are industrial examples where large

[1]Constraint based systems that handle also real numbers exist.

orders have been lost because of certain variants being prohibited due to unexplained numerical constants in the knowledge base. In a situation pressed for time it might not be possible to investigate whether obstructive data could be changed or not, unless an explanation of its origin is given.

An industrial system represent a considerable investment and is intended to be used and maintained for many years., meaning that it must be possible for new personnel to continue the work of the original knowledge engineer(s) and programmers. It is obvious that the knowledge base must be easily human readable, otherwise the system will over time become a black box which no one understands or can control. Knowing *what* data, rules and methods that are applied is however not enough, there must also be references to the background describing *why* this knowledge should be applied. These references could be in the form of references to comments, standards, design manuals or experimental data. The importance is due to the need to update the knowledge base if changes occur in e.g. technology or customer preferences. Another important reason to spend resources on background knowledge is when a new product family is to be created. It is then very likely that large parts of existing code could be reused, but only of course, if the conditions under which the coded knowledge are applicable. Also this requires an explanation of why certain knowledge is used and what the limits of its application are.

The big challenge is to keep program and documentation synchronised over time. It requires an almost utopian self discipline to remember that every small correction or change to the code must be documented. Some sort of automated prompter or restriction for load of updated code might be well advised.

The necessity of exhaustive documentation is thus clarified. There is however a less defensive argument to spend resources on documentation: The possibility to use the knowledge base of the DA system as compilation of the corporate knowledge possessed by the company. The stringency of the computer implementation forces this documentation to be very detailed and explicit and thus provides a very exhaustive design manual.

8.5.2 Formal Methods for Documentation

The method for documentation must be tailored to each specific company and its products. There are however a few formal methods that should be mentioned as examples to study. A Product Variant Master, see Sect. 3.3.2 is a natural starting point and defines the product structure and possible variants. To the branches of this tree structure the rules governing the configuration are attached. In principal a standardised form is set up for each rule or group of rules that are to be documented.

One such system of forms has been established within an EU program for the aircraft industry (Stokes 2001) and is based on a set of cards called ICARE. This acronym stands for **I**llustration (geometry), **C**onstraints, **A**ctivity (process, control), **R**ules, **E**ntity (product structure, geometry). The system is extensive and also computerised and is intended for large scale projects.

A somewhat less extensive system which seems to be much used is based on CRC cards, which is an acronym formed from **C**lass, **R**esponsibility and **C**ollaboration, see (Hvam et al. 2008). The CRC cards specifies meta data like "Author", "Date", and a specification of the task to be carried out. A graphical illustration of the part should normally be included. Then the names of the input parameters, rules to be applied and the names of the resulting output variables are specified. References to the origin of the knowledge base could be given as Collaborations. One natural way to arrange the CRC cards is to structure the knowledge similarly to the product structure. The position of the card in the chosen structure, is defined by specifying "part of" and "consists of". If an object oriented structure is used, then the names of superclass and subclass are specified. When the complete knowledge used has been stored on CRC cards a coherent and well structured documentation has been established.

A third, more generic option should be mentioned. A standard, called **U**nified **M**odeling **L**anguage, UML originally developed for software modelling, has become popular also for other object oriented modelling tasks. The method is well suited to model knowledge and the resulting information model is well adapted for implementation in a relational database. One attractive possibility is to arrange the information model like a design manual and set up links between the product structure and information model. For design rules in the form of algorithms there are possibilities to use a form of documentation that is both human readable and computer executable, thus guaranteeing that executable code and documentation is always synchronised. This technology is called literate programming, see (Knuth 1984), and commercial programs like MathCAD and MatLab belong to this category.

8.6 Aspects on Security

The knowledge base of a design system is a compilation of vital corporate knowledge that is concentrated, structured and stored at a single location. Consequently, there is a risk of proliferation of knowledge to unauthorized persons. This is an objection that is often raised among potential users of design automation, but is seldom a concern for companies having experience from the technology. The main reason is probably that even if the knowledge is sensitive, it is a part of a specific industrial system and it is difficult to use it outside its normal context. If, despite of this, a risk is perceived, there are a number of security measures that can be taken:

- Avoid running a design system on computers with Internet connection and remove all portable facilities for copying
- Lock the system computer and/or store it overnight in a safe room
- Use hardware locks and passwords

- Divide the design system into parts that can not be executed separately and run these on different computers
- Separate programs from data files
- Store program back-ups and documentation in a safety vault

All these actions complicate development and use of system and hence, must be carefully balanced against the perceived risk.

8.7 Knowledge Quality

This book has discussed how to capture, represent and process knowledge. The usefulness of doing this depends entirely on the quality of knowledge—an efficient design system is of no use unless the knowledge implemented in it is of high quality., i.e. correct, complete, consistent, valid, documented and well structured. The rules and methods implemented should reflect current best knowledge and best practice, which requires regular updating of the knowledge base to assure that new insights are paid attention to.

Over time new product models and requirements are introduced and many characteristics of the products gradually change. When this is the case it is important to reassess design rules especially approximations that are applicable only within a certain range to verify that these are still valid and acceptable. Also, areas with known weaknesses might need concentrated research efforts to improve the insights available. This quality assurance work is made much easier if the knowledge documentation is well structured and transparent. A neglected knowledge base will over time become something of a digital dustbin, but well cared for it could become a gold mine of relevant and accessible corporate knowledge—the company's most valuable asset (Fig. 8.13).

Fig. 8.13 Gold mine or dust-bin?

References

Department of Defense: Systems Engineering Fundamentals. Defence Acquisition University Press, USA (2001)
Eppinger, S., Browning, T.: DSM. MIT Press, Boston (2012)
Flower, M.: Domain. Specific Languages. Addison-Wesley, Boston (2011)
Hvam, L., Mortensen, N.H., Riis, J.: Product Customization. Springer, Heidelberg (2008)
Johansson, J.: Automated Computer Systems for Manufacturability Analysis and Tooling Design. PhD thesis, Chalmers Technical University. ISBN 978-91-7385-510-5 (2011)
Knuth, D.: Literate Programming. The Computer Journal. Wiley, London (1984)
Ryhanen, E.: A business strategy based on automated design and manufacture. Notes from conference design for product variety. JTH, Jönköping June 2004
Stokes, M.: Managing Engineering Knowledge. Professional Engineering Publication Ltd., London, 2001LiTH (2001)

Appendix A
Industrial DA Systems in Production or Prototyping

How do the theories and methods discussed in this book agree with experiences from real world applications? A systematic and exhaustive answer to the question would require a much larger investigation than what is possible to present here. The intention with this Appendix is instead to give the reader real, working examples as background information when studying the methods presented. This helps to better understand what applications that could be efficiently solved using DA, how the different technologies presented are used in practice and what obstacles that may arise and different ways to overcome these. Industrial systems for design automation have been used at least since the 1970s, and this Appendix will give examples from a wide range of different applications and different technologies.

Often large scale industrial systems are close to the core of a company's operations and therefore surrounded by a degree of secrecy. The examples described in this Appendix are therefore of a varying completeness and also belong to different periods in time. Some of the systems are large, but straightforward industrial applications, while others represent small prototype systems aimed at more difficult problems that stretches the boundaries of the technology. The systems have different development histories ranging from management initiatives with adequate planning and properly allocated resources to what could be seen as "skunk works" where individuals or small groups have seen opportunities for drastic improvements of efficiency and acted spontaneously on their own initiative. Design automation is far from a late innovation. More than 50 years ago one article (Fielding 1965) was published that offers an interesting historical perspective, showing that the questions raised in the mid 1960s are still highly relevant.

The majority of Industrial systems for DA are aimed at product configuration, possibly complemented by parametric modelling of components. Engineering design is however a topic that embraces many aspects of technology and future DA systems will be capable of handling more demanding tasks in redesign, which might require more design freedom than straightforward configuration and

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4

parametric design. The list therefore also includes applications that represent very different solution approaches exemplifying virtually all methods discussed in this book.

Essentially it is left to the reader to form his/her impression of industrial DA systems out of the list of applications compiled below, but some reflexions or conclusions for future work are given:

- Setting up a DA system is a demanding task—do not expect the first attempt to be successful. Build small prototypes and test. Systems often need to be developed from scratch more than once until the right approach is found
- There is decidedly not any standard solution that fits all tasks
- It is beneficial to develop a new product and its associated DA system simultaneously
- In a development project it is not unusual that the DA program development cost is of comparable size to the cost of development of the product itself.
- Take the opportunity to abolish product documentation (drawings, BOM, control instruction) that might become unnecessary when the new system is put into production
- Use geometrical functionality of the CAD system, but limit use to standard functions that will always be available in new versions or new systems
- Vendor dependence and operational life of commercial software could become a problem. Consider at an early stage how the system could be migrated should that become necessary
- Maintenance becomes important over time and should be taken into account when building the system
- Documentation essential for long and safe operational life
- No examples found where companies, once having established a DA system, later have abolished it and gone back to manual operations.

A.1 Electrical Power Transformers

Translated and adapted from Sunnersjö (1993) with the permission of Industrilitteratur AB (now lamanica AB)

This example describes an application in production between mid 1970s to mid 1990s. The system is of a traditional type and all functions, including drafting, is integrated in the procedural code. It represents a technology for implementation that is now outdated, but the example is instructive because its full life span is documented including the effects the system had on the design process of the company (Fig. A.1).

The product is based entirely on mature technology and the design and production processes are order based. On average 1.3 units were delivered per variant of medium and large size transformers. This places the product at the top right corner of Fig. 4.1, i.e. an application with good potential to be profitable.

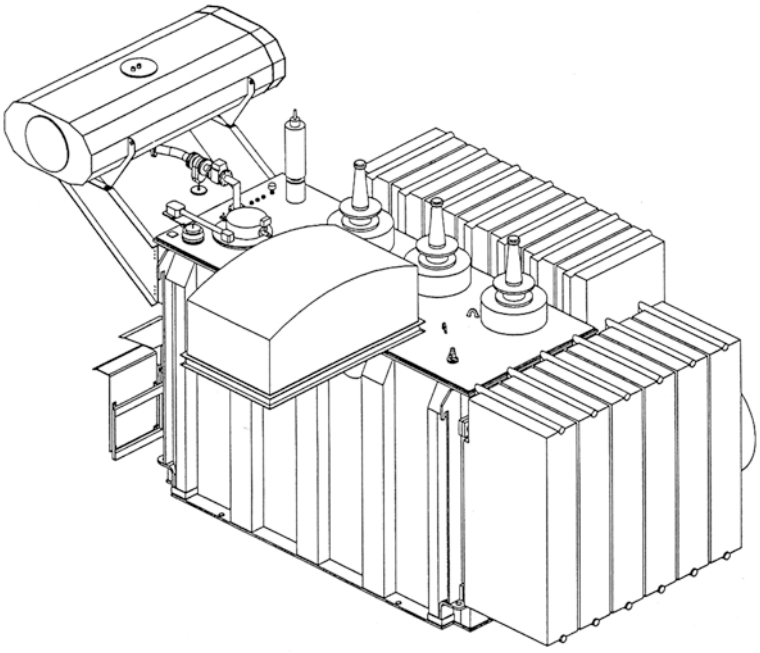


Fig. A.1 Three-dimensional view of transformer unit. From Sunnersjö (1993)

The programs are written in Fortran and comprises about 250,000 lines of code and about 20 man-years invested in programming. About the same effort was spent on streamlining the product family for efficient variant forming. A pilot study of using macro programs for 2D CAD was carried out but found to be less cost-effective.

A complete set of drawings amounts to about 150 manufacturing drawings per order. This means about 12,000 drawings per year and in addition a few hundred quotations including a small set of drawings and cost calculations each. Approximately 95 % of the documents could be used directly in production, while the remainder required some manual modifications. The productivity gain compared to the previous, manual process was about 10:1.

The system is built around a basic product design from which variants are created by configuration of parametric components. When the system went into production about 90 % of enquiries could be processed automatically, the remaining 10 % required manual design. When the system was closed down after 20 years of operation, only about 30 % of small transformers and 70 % of medium sized transformers could be processed automatically. It appears that when market and technology changed, it proved difficult to update the programs quickly enough to implement required changes and thus, an increasing proportion of the design work again had to be taken care of manually. This highlights a significant drawback with the procedural standard program used. In the late 1990s the original system was replaced by a parametric solid modeller with internal, procedural macros.

A.2 Cutting Tool Holder

Name of system: "Tailor Made" (Fig. A.2)

The company manufactures for instance tooling for milling, drilling and turning for the manufacturing industry world-wide. The products are supplied from stock or customised on demand. It is business strategy to use a design automation system both for order based design and to generate products in size ranges to be delivered from stock.

When an inquiry is received the system makes preliminary drawings and estimates of cost and delivery time within 24 h compared to the previous lead time of three to four weeks. If the offer results in a customer order the detailed design with drawings and production planning including NC preparation and control instructions is carried out. The tailor-made component is delivered within one to two weeks compared to the previous delivery time of about ten weeks. Administrative costs are almost completely eliminated.

Development of the system was initiated in 1984 and it has an interesting history spanning 30 years. The DA-system is developed by a team of about 30 persons working in parallel with the product designers. When a new product development project is started a parallel DA project is also started so that when the new product is available it can immediately be generated in customised variants.

A first version based on parametric 3D CAD (wireframe) with macro programming was taken into production from 1985. This was replaced in 1994 with a commercial rule based system integrated through APIs with a parametric 3D solid model. This system has in excess of 14,000 rules and 11,000 methods and is a vital part for the operations of the company. When the software vendor stopped supporting the rule based system the company decided to develop future software in-house. An interim system was developed and put into production in 2006, while a more permanent software was developed from scratch and put into production in

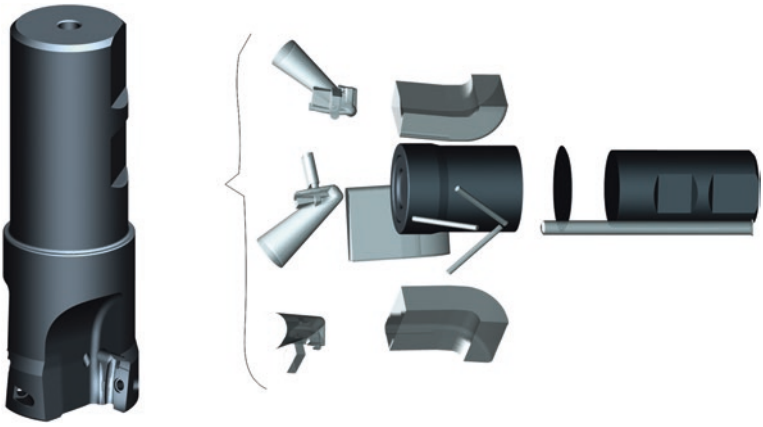


Fig. A.2 Milling head (*left*) assembled from parametric parts (*right*). Courtesy of Sandvik Coromant AB

2011. This program is written in Java, uses open source code, and implements a domain specific language (DSL). DA logic is expressed in the DSL using declarative backward chaining rules. The DSL also have high level interfaces with the CAD system which enforces that company standards for creation of CAD models are followed.

All in all four different software platforms have been, and still are, used in production which represents a significant maintenance effort. This experience highlights the importance of:

- A well-defined problem domain
- Using standardised CAD methods
- Using conventional programming languages
- Good scalability
- Planning for maintenance
- Methods for verification and testing
- Documentation of design knowledge used

A.3 Ventilation Systems

Name of system: Selection configurator (Fig. A.3)

The company designs and produces custom built ventilation systems in an order based process. Product configuration, parametric component design and cost calculation has been done by an automatic configurator since the 1990s. The system includes an extensive knowledge base with technical, legislative, customer and regional preferences. When an order is placed the product is specified by an alpha-numeric code (about 10 groups each with 30 digits). The configurator is used by sales people and installation technicians.

After order the delivery process starts by generating the detailed product definition. A small number of 2D CAD drawings are created for assembly instructions and for export to building drawings. Otherwise no drawings are produced

Fig. A.3 Customised ventilation units. Courtesy of Fläkt Woods Group

and furthermore, no product definition needs to be saved after delivery—only input data and program version. If a spare part should be needed the design is regenerated.

Subsequent to the detail design process, production planning and NC preparation is carried out automatically and the stations at the production line receive instructions for each order individually. The production planning system processes about 100 orders per day. Approximately 30 % of these need minor manual modifications. The system has passed through several generations and been coded in different programming languages over the years. It is now run over the internet using a procedural code in VB and a relational database. It is estimated that in excess of 100,000 man-hours have been spent building and maintaining the system.

A.4 Roof Rack Attachments

Name of system: Tracks

Adapted from Johansson and Cederfeldt (2012) and (Johansson (2012) in cooperation with Thule Sweden AB

The company manufactures roof racks for motor cars and sells world-wide (Fig. A.4). For cars without fix-points the roof rack is attached to the roof with a system of fixturing brackets and rubber mounting pads. Each car model requires individually designed attachments due to different roof curvatures. Since the roof rack is a safety related component calculations or tests of its ability to meet crash test requirements has to be made and approved by national road safety authorities.

About 150–200 new car models are launched each year and the company collects measurements of roof profiles for about thirty new fixture kits on international motor car fairs and at other opportunities. These profile measurements are used to design new brackets and rubber pads. From a business perspective it is vital to offer car buyers roof racks as soon as a new model is launched. A customer who buys a roof rack is likely to later also by accessories like roof box, bicycle holder and so on.

The company has about 1500 variants of fixturing brackets either in stock or as drawings, tooling and so on. Rather than designing new variants for each new car model the idea was to reuse similar, existing and proven solutions. This would save time and money from design, calculation, tooling, testing and approval of new variants. A system based on Case Based Reasoning was initiated.

The fixturing brackets were divided into natural classes, see Fig. 7.2 in Sect. 7.1. Within each class the specifications of the new design is compared to the designs stored in a case base and rated for how well they agree. The variants that are closest to the goal geometry are retrieved by the DA system and displayed simultaneously with the the outline of the roof. The necessity for the graphic display became obvious with practical use. If one of the retrieved cases are very close

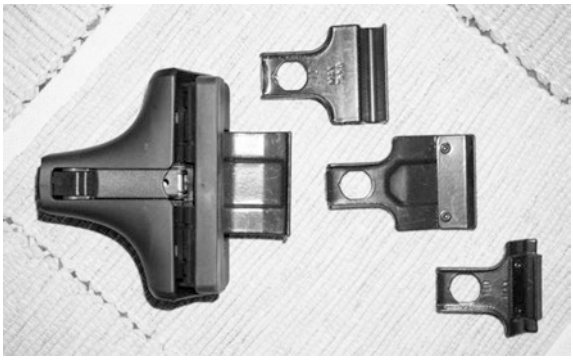


Fig. A.4 Mounting system for roof rack manufactured by Thule Sweden AB

to the goal fit, this will be used for the new application. If the best retrieved variant is close but not quite acceptable the required modifications are carried out and the component stored in the case base. Probably the component can be produced without new tooling or new tests. If no close variant is found a new bracket design is carried out automatically using the goal geometry as specification. The new design is stored in the case base.

A corresponding procedure is carried out to select best match among the 400 stored rubber pads. This presents a more difficult problem of shape matching which is solved by analysis of the clearances arising between roof and existing rubber pads in a sequence of positions. This simulation and analysis is carried out by the CAD system.

The system took about one man-year to develop in procedural VB code during which time several prototypes were tried and discarded. The shape matching is computationally heavy and the first working prototype took 45 minutes to scan the case base. Execution time could later be reduced to less than three minutes, which drastically changed the usefulness of the system. The system was put into full use in 2012 and has since been used for virtually all new attachment kits. The design time for new variants has been decreased by about 40 % (two weeks), tooling costs is reduced by about 50 %, quality assurance improved and total lead time was reduced in excess of 10 %.

A.5 Plate Heat Exchanger

System name: Cross-roads (Fig. A.5). For further details see (Rask and Sunnersjö 2012)

The company designs and manufactures plate heat exchangers for different liquids in a variety of sizes, pressures and user specifications. The heat exchanger consists of a stack of plates, each with pressed channels that form a system for

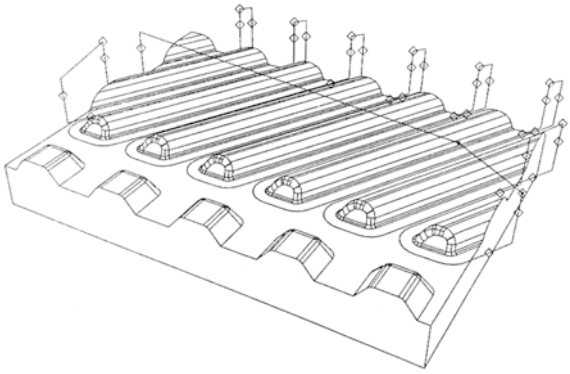


Fig. A.5 Press die for plate in heat exchanger

process fluids. The patterns and dimensions of the channels are of critical importance for the performance of the heat exchanger. The channel design is the focus of the DA-system. The design process includes engineering design, stress analysis, fluid flow analysis and manufacturing processes. The motives for automation are twofold, to assure that all designs are made according to defined instructions and to optimise heat transfer.

Investigating the steps of the design process and coding the DA-system took about four man-years. A DSM was set up which showed a box with cyclic dependencies, see Fig. 5.1 in Chap. 5. The sub-tasks were set up in an order that has evolved over many years of practical experience. When the coupled part of the DSM was broken down to parameter level and partitioned it turned out that two boxes were present, each representing an optimisation problem. The objective was to the maximise channel cross section area under constraints from strength and production requirements.

Cross-Roads was programmed in a rule based program with inference engine feeding design variables to a parametric solid modeller. The system designs the heat transfer surfaces of the plates as well as the press tools used in manufacture. The system was used for all new plate design world-wide between 1995 and about 2005. When the rule based software was no longer supported the system was rebuilt in a procedural, algorithmic programming tool (Fig. A.6).

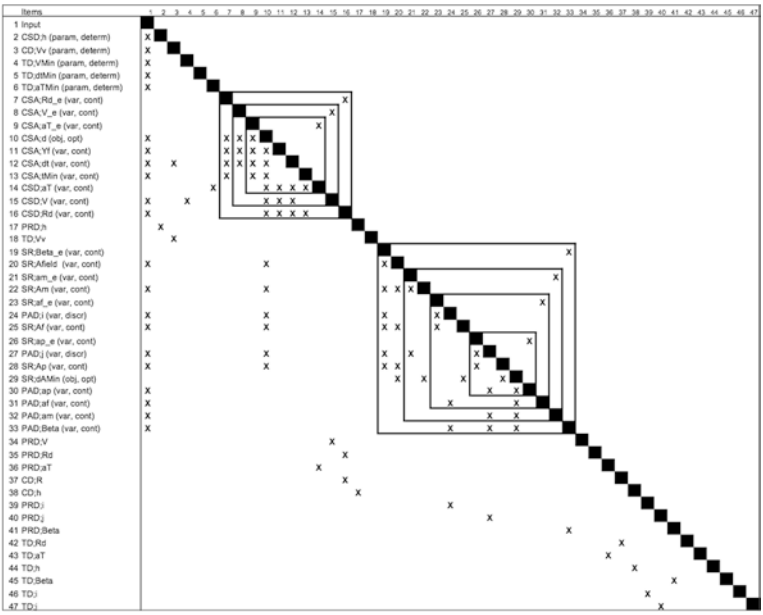


Fig. A.6 DSM for flow channel design. *Top box* refer to optimisation of channel cross section, *bottom box* refers to optimisation of channel pattern. Courtesy of Alfa Laval Thermal AB

A.6 Bulkhead Stiffeners for Submarine

System name; CORP

For further details see Sunnersjö et al. (2004) (Fig. A.7)

Bulkheads in submarines are reinforced with welded stiffeners, which have the cross-section of a T-beam. Both plate and stiffeners are made of high tensile strength steel of massive dimensions which require expensive and time consuming welding procedures. There are thus strong motives to design the complete bulkhead as light as possible. Having the same principal design the stiffeners are parametrically adapted to their respective length and position on the plate. There are two business motives to automate this design process: Quick response to customer enquiries and finding the most cost-effective balance between number of stiffeners and dimensions of stiffeners.

A thorough investigation into the calculations of stiffener dimensions resulted in a DSM with a hierarchical data flow. The calculations were made in an algorithmic and literate (self-documenting) language thereby providing good and automatically updated documentation. The calculations were quite extensive amounting to about 90 pages of algorithms. The pages were arranged as a design manual where each subtask formed a section which was readable but at the same time had the role of a knowledge object with inputs/outputs (Fig. A.8).

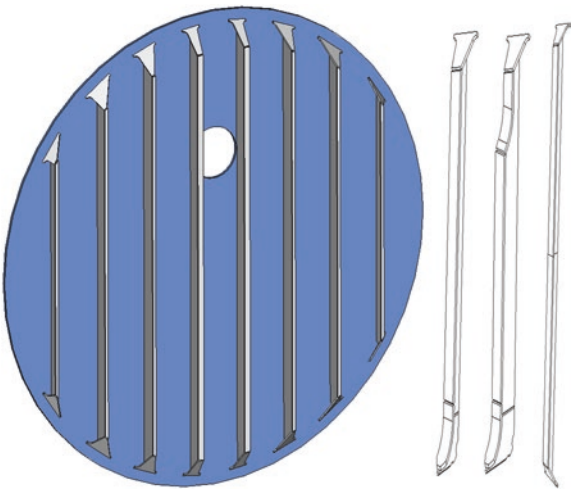


Fig. A.7 Bulkhead with parametric stiffeners

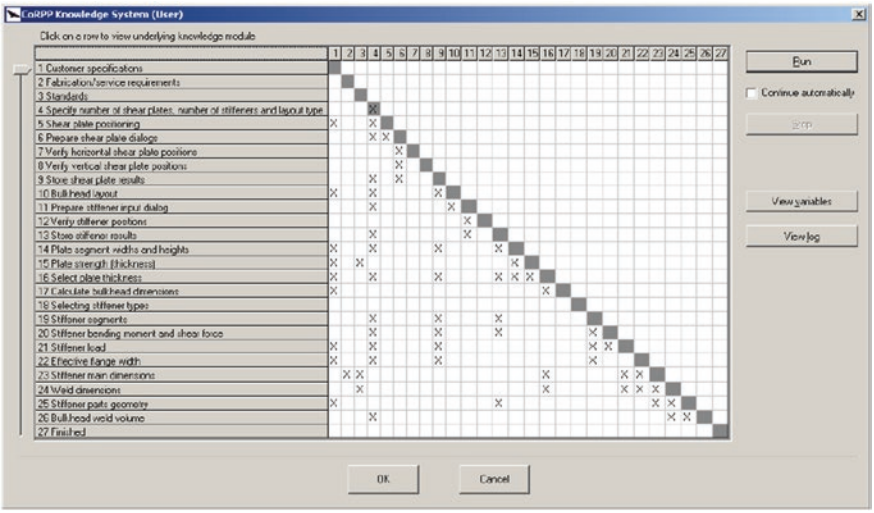


Fig. A.8 Static DSM with sequential information flow acts as a work-flow manager

The static DSM acted as a workflow manager and activated stepwise processing where the variables from previous steps was used as input for subsequent steps. As the processing proceeds down the DSM, eventually all design variables were determined and stored in the database. For each stiffener an optimisation process was executed to determine its optimal proportions. All design variables are subsequently fed to a parametric solid modeller for graphic display, generating drawings and to carry out stress calculations.

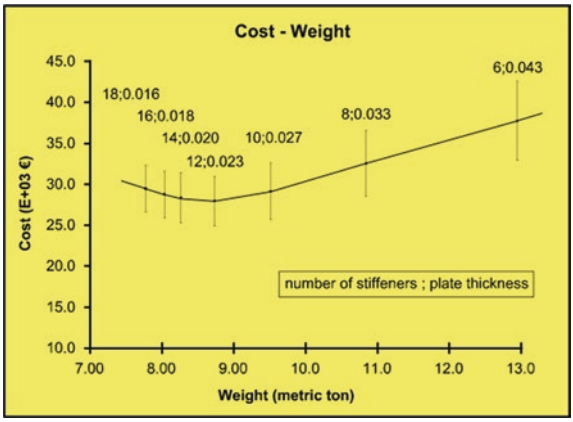


Fig. A.9 Cost and weight of bulkhead as a function of number of stiffeners. For this case optimum occurs for twelve stiffeners with 23 mm plate thickness

From the database information was extracted for process planning and related cost estimates. A library of generic, standard processes for the types of operations to be used, were instantiated with the current variable values and time and cost for each component and feature were determined. These are accumulated and added to the costs for material which results in the total cost. To find the optimal combination of number of stiffeners and plate thicknesses, calculations are carried out for a suitable range, e.g. between six and eighteen stiffeners, as in Fig. A.9.

A.7 Rolling Element Bearings and Tooling

Translated and adapted from Sunnersjö (1993) with the permission of Industrilitteratur AB (now Lamanica AB)

Rolling element bearings are made and stocked in size ranges, where inner and outer diameters are driving dimensions. The differences between sizes are not simply a matter of scaling dimensions in proportion. Instead, each size needs to be recalculated and redesigned from basic design knowledge.

A rolling element bearing is typically defined by 400–700 design variables. The system also designs blanks and tooling which are defined by about 500 variables. Hence, for each dimension in the size range more than 1000 variables will need to be calculated making this application a strong case for automation.

The system was developed in the mid 80s as a procedural program (Fortran) feeding a 2D parametric CAD system. It was used daily for about 15 years with revisions at least once a year. In mid 1990s it was replaced by a parametric solid modeller with internal macros as well as external programs for computations. The system had an estimated productivity gain of 1:40.

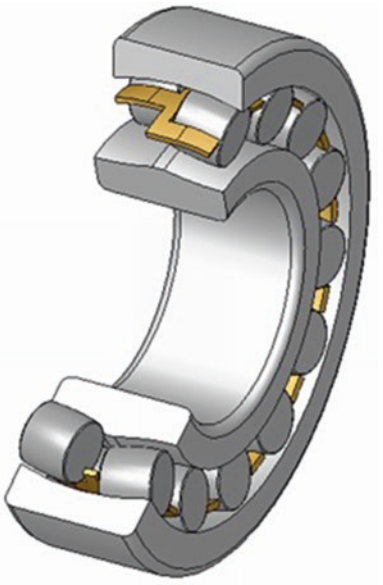


Fig. A.10 Double row spherical roller bearing

A.8 Seat Heaters for Motor Cars

Name of system: "Wire-layout"

For more detailed information see Johansson and Elgh (2013)

Seat heaters consist of a foam carrier on which resistance wire is glued in an evenly distributed pattern. When required electrical current is led through the wire thus producing an increased seat temperature. The company sells to the motor car industry worldwide. Typically 75 new variants are put into production each year requiring four to six weeks of design work within a total lead time of 12 months from order to production.

A very large number of enquiries are received each year, and from a business perspective prompt and accurate quotations are of vital importance. Although seat heaters are a seemingly simple product about 160 design variables are required to define the design. To improve the capacity for quotations a design system for a preliminary wire-layout was developed. A time and cost calculation module was added to make it possible to use the preliminary design for cost calculations (Fig. A.11).

A second purpose of the system was to determine which wire type that will be most cost effective. Reaching the specified heat effect will depend on the wire type, the length of the wire and the pattern. To find the best alternative for a specific application a range of suitable wires are run on the system, and the

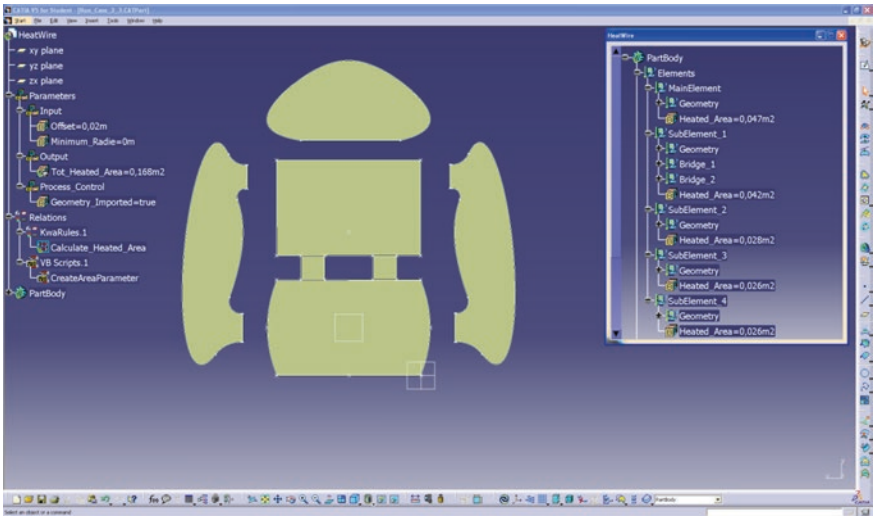


Fig. A.11 History tree with design rules inserted and lay-out (*top*). Generative system with hierarchical data flow. DSM for design process (*bottom*). See also Fig. 3.13 in Sect. 3.4. Courtesy of Kongsberg Automotive AB

preliminary designs are evaluated and the most favourable cost is used for quotation. If the offer results in an order the preliminary design is reviewed and can usually be somewhat improved manually. The system, which could be classified as generative, uses a ruled based solver with inference engine that interacts via an API with a parametric CAD system with macro-programmed instructions. The workflow is sequential with two exceptions which are solved by manual interaction. Generation of a wire layout takes a few minutes on a lap-top. The system is at the time of writing a prototype, not a production system.

A.9 Tool Set for Aluminium Tube Bending

Adapted from Johansson (2011) in cooperation with Sapa Group. Name of system: "Bend-IT" (Fig. A.12)

Aluminum tubing is formed using a draw bending process. The required tool set consists of: Form die, pressure die, clamp die, mandrel and wiper. In most cases all or several of these tools must be designed and custom built for each new order. To save lead time there are good motives to automate this design process.

The manufactured parts will always deviate somewhat from the nominal geometry, which is acceptable as long as these deviations are within specifications. The tool design together with the setting of process parameters govern the outcome of the bending process, particularly the developed length and the centreline radius

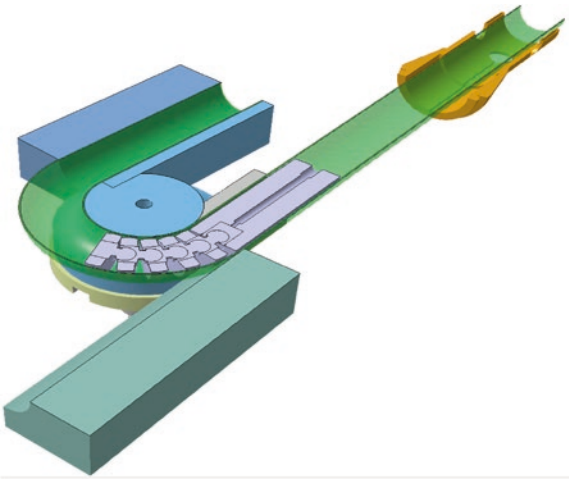


Fig. A.12 Tool set and tube in bending operation

of the bend. Undesired effects like wrinkling, ovality of correction and decreased wall thickness must be kept within specifications. The required bending moment and angle needs to be estimated and whether or not a mandrel is needed. If this is the case a suitable type of mandrel should be selected.

The knowledge used to plan this process and design the tools that will produce parts within accepted tolerances is a mixture of heuristics and mathematical formulas. This is represented in a rule based design system having an inference machine and feeding a 3D parametric CAD system. The system is based on a knowledge object architecture giving a highly modular structure, which later proved very valuable when the system was expanded to profile bending, multi bend operations and preparations for FEM simulations. The system is at present a prototype, not in production.

A.10 Aero Engine Diffusor

The company designs and manufactures components and subsystems for aero engines. Preliminary design for quotation is a critical business factor where automation enables fast and correct response to enquiries. The company also benefits significantly from being able to generate and evaluate many lay-outs to find the most cost efficient design concept (Fig. A.13).

First the extent and desired variation is defined, wherafter a parametric 3D CAD with an internal, rule based solver, is run with a high degree of automation. A range of alternative configurations is created as solid models as well as associated shell models that are automatically pre-processed and analysed using Finitie Element solvers.

Structural integrity of each concepts is evaluated, including eigenfrequences and modes, stresses, distortions and more. The application further generate core input for cost assessment and manufacturing key parameters.

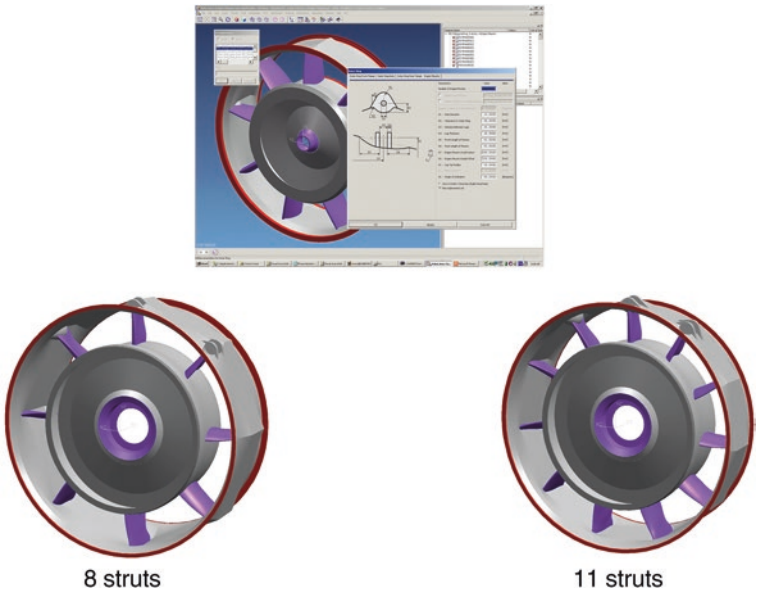


Fig. A.13 Design system GUI (*top*) and two diffusor variants to be evaluated. Courtesy of GKN Aerospace Sweden AB

The system has been in use from around 2006, mainly in the sales and early concept design process. As an example the system created, defined and evaluated 14 different lay-outs in two weeks, something that far exceeds what is possible with manual manipulation of a parametric modeller. The system has proved to be very valuable also in discussions with customers during the sales process. The company uses several other generative design automation systems in their sales and design process.

References

Cederfeldt, M.: Planning design automation. Ph.D. Thesis, Chalmers University of Technology, Göteborg, Sweden (2007)

Fielding, F.: Cost value of design automation. In: Proceedings of Annual ACM IEEE Design Automation Conference, ACM Press, New York (1965)

Isaksson, O.: KBE and CAD for concurrent engineering applications. Lecture notes from conference "design Automation", May 2007 at Fläkt Woods Group, Jönköping, Sweden (2007)

Johansson, J.: Combining case based reasoning and shape matching. In: Proceedings of ASME DETC2012-70631, Chicago (2012)

Johansson, J., Cederfeldt, M.: Interactive case based reasoning through visual representation. In: Proceedings of Design 2012, Dubrovnik, Croatia (2012)

Johansson, J., Elgh, F.: How to successful implement automated engineering design systems. ISPE CE (2013)

Nilsson, In: Conference of the Association for product modeling, Aarhus Denmark. Department of operations management, DTU, Lyngby, Denmark (2010)

Rask, I., Sunnersjö, S.: In Design structure matrix methods and applications by Eppinger, S., Browning, T., MIT Press 2012, pp 216–221 (2012)
Sunnersjö, S., Cederfeldt, M., Elgh, F., Rask, I.: A transparent design system for iterative product development. J. Comput. Inf. Sci. Eng. **6**(3) (2006)
Sunnersjö, S.: Förberedd konstruktion. Swedish association of manufacturing companies VI, Industrilitteratur V020006 (1993)

Appendix B
Exercises

Some of the cases of applications in this book have been adapted and simplified to serve as exercises. They require implementation in traditional programming languages like C or VB and access to commercial software for mathematics and CAD. The details of these implementations are not meaningful to reproduce here as they will be program and version dependent. Instead the reader is encouraged to attempt to solve the tasks with available and familiar software.

- Tutorial one demonstrates parametric modelling of a standardised beam cross section which is driven by a template. It is also demonstrated how an external calculation program interacts with the CAD model through the template.
- Tutorial two is an example of a design automation problem that require a generative solution model. Knowledge is stored both as part of the CAD model and in external macros that generate the product model.
- Tutorial three is an exercise in parametric design with all knowledge stored as a part of the CAD model. The execution order is shown to change automatically when the design variable asked for is changed. It is also shown how new rules can be added without any side effects on existing code.
- Tutorial four concerns the planning of a design automation system by applying the methods described in the book. It serves as a summing up of the guide lines presented

Tutorial 1: Parametric design using templates
By Mikael Cederfeldt

Use a parametric CAD system to create a model allowing for variant design of the two beam types, I and HEA, depicted below (Fig. B.1). Drive the CAD system from a template that can either use the beam standard size as input or where all parameters are explicitly defined. Create only one geometry model (with on/off features) that can be parametrically configured into the variants shown in the beam type standards tables. Add a short program module that uses the variables to calculate e.g. maximum tensile stress for a given bending moment.

S. Sunnersjö, *Intelligent Computer Systems in Engineering Design*,
Studies in Systems, Decision and Control 51, DOI 10.1007/978-3-319-28125-4

Tutorial 2: Generative design using APIs
By Fredrik Elgh

Problem description

The exercise is taken from a project which included the development of an design automation system. The scope of the system was to generate variant designs of heating elements for car seat heaters based on different customer specifications and seat geometries. A heating element consists of a carrier material, a wire and a connecting cable. The wire is laid out and glued in a pattern of sinusoidal loops between the two layers of carriers, see Fig. B.1. The pattern is generated essentially using heuristic knowledge.

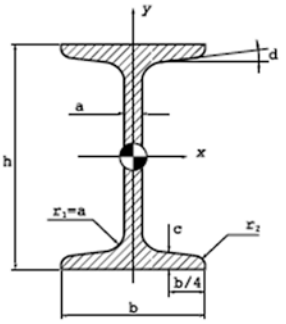
The objectives with the DA-system were to cut quotation lead-time, allow for evaluation of different design alternatives, quality-validate the design process, capture design knowledge, ensure producibility and provide design documentation.

Learning outcome

The purpose of this tutorial is to illustrate the concept of Generative Design. Most design automation systems assumes that a predefined generic model has been set up. This model must contain all possible variations of dimensions and topology, which are then instantiated to create one particular variant. There are however situations when it is not possible to set up such a generic model (e.g. the topological variation is too large) and the product geometry is instead generated under program control. The task presented is such a case. The solution to the original problem with real seat geometry was shown in Fig. 3.13, while the tutorial addresses a simplified geometry. It is a fairly demanding task, but instructive on macro programming of CAD systems. It has been solved using Catia Knowledgeware with macros written in VBA. Most programmable, parametric 3D solid modellers should be able to handle the problem in a similar way.

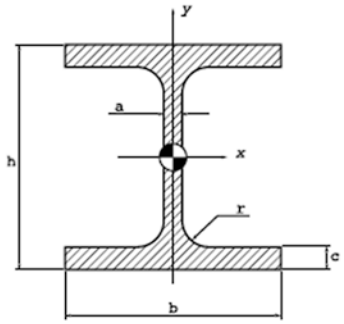
Implementation task

In this tutorial the problem is reduced to a rectangular carrier element where height and length can vary continuously. The objective is to define the centre-line of the wire-pattern so that the heat is high and evenly distributed across the element. This implies that the objective is to fill the element with so much wire as possible, i.e. try to maximize the centre-line length. The output is the length of the centre-line and the winding pattern (Figs. B.2, B.3 and B.4). There are two constraints that cannot be violated: There must be (1) a clearance between the edge of the element and the centre-line, and (2) a clearance between the wireloops within the pattern.



I-Beam

Size	h [mm]	b [mm]	a [mm]	c [mm]	r_2 [mm]	A [mm^2]	Weight [kg/m]	I_x [cm^4]	W_x [cm^3]	i_x [cm]	I_y [cm^4]	W_y [cm^3]	i_y [cm]
I 80	80	42	3,9	5,9	2,3	7,58	6,0	77,8	19,5	3,20	6,29	3,00	0,91
I 100	100	50	4,5	6,8	2,7	10,6	8,3	171	34,2	4,01	12,2	4,88	1,07
I 120	120	58	5,1	7,7	3,1	14,2	11,2	328	54,7	4,81	21,5	7,41	1,23
I 200	200	90	7,5	11,3	4,5	33,5	26,3	2140	214	8,00	117	26,0	1,87
I 260	260	113	9,4	14,1	5,6	53,4	41,9	5740	442	10,4	288	51,0	2,32



HEA-Beam

Size	h [mm]	b [mm]	a [mm]	c [mm]	r [mm]	A [mm^2]	Weight [kg/m]	I_x [cm^4]	W_x [cm^3]	i_x [cm]	I_y [cm^4]	W_y [cm^3]	i_y [cm]
HE 100A	96	100	5	8	12	21,2	16,7	349	73	4,06	134	27	2,52
HE 120A	114	120	5	8	12	25,3	19,9	606	106	4,89	231	38	3,02
HE 160A	152	160	6	9	15	38,5	30,4	1670	220	6,57	616	77	3,98
HE 180A	171	180	6	9,5	15	45,3	35,5	2510	294	7,45	925	103	4,52
HE 220A	210	220	7	11	18	64,3	50,5	5410	515	9,17	1950	178	5,51

Fig. B.1 Standard I and HEA beam cross sections

Tutorial 3: Design rules as part of CAD model
by Staffan Sunnersjö and Roland Stolt

Description of exercise

The automatic balancer and its working principles are described in Sect. 5.1.

The purpose of this tutorial is to create a system for parametric design using declarative rules or formulas being a part of the CAD model. It will be

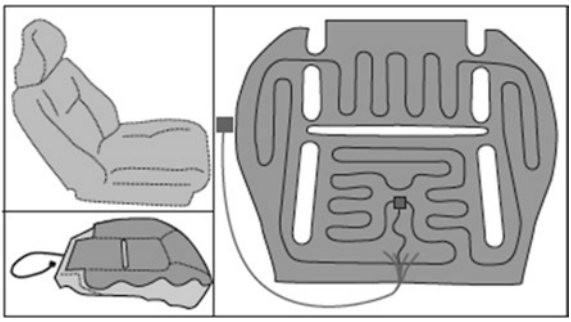


Fig. B.2 *Upper left* A car seat with heating elements in the cushion and backrest. *Lower left* A cushion element glued to the seat foam. *On the right* A cushion element showing the heating wire, the thermostat and the connection cable between two layers of carrier material

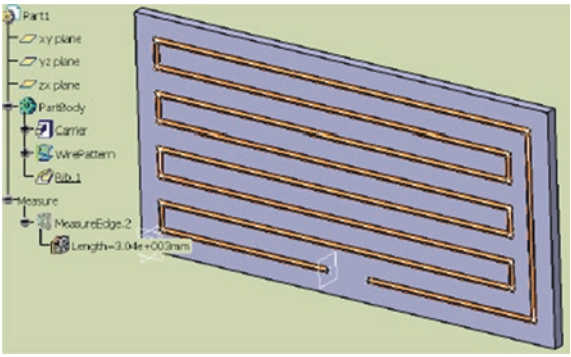


Fig. B.3 Winding pattern and a measure of the centre-line's total length

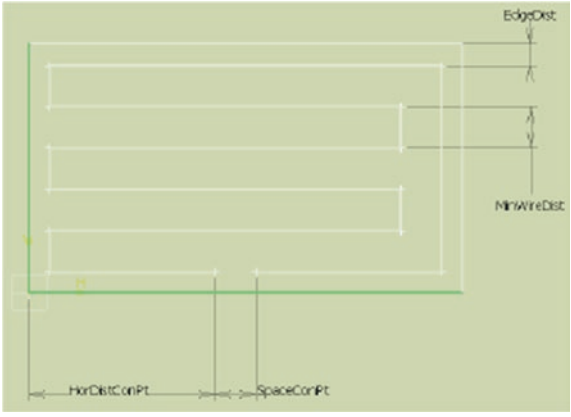
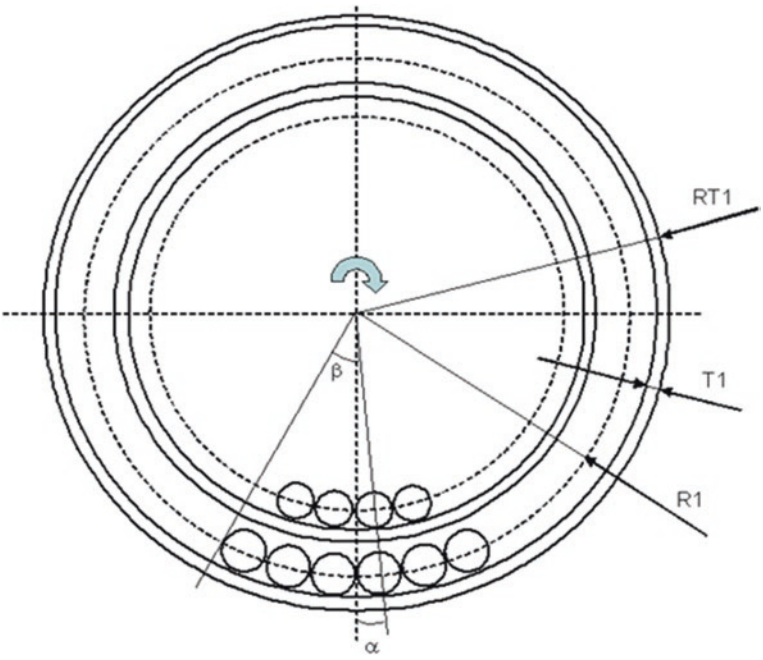


Fig. B.4 Main constraints of the heating wire centerline

demonstrated that parameters can be changed dynamically and that this results in different solution sequences. It will also be demonstrated how new rules can be added to the rule base and automatically being taken into account. For simplicity, assume that the balancer has one track and the number of balls is equal to two. The example was implemented directly in Catia Knowledgeware, but several other CAD systems have comparable functionality.

Specified parameters (input)		
Φ_1	Diameter of balls	0.02
Ω	Rotational speed (rev/min)	1200
N	No of balls	2
ρ_B	Density ball mtrl	7800
UBM	Unbalance moment	0.02
σ_y	Yield stress of ring	15E6
Intermediate design variables		
BM	Mass of ball in track	
α	Angle ball	
fc	Centrifugal force	
BMR1	Balancing moment/ball	
σ_d	Yield stress dimensioning	
Design variables (output)		
RT1	Outer radius of ring	
HT1	Wall height	
T1	Wall thickness	



Design rules

The design of the balancer is subject to the relations below. These assignments are to be programmed as design rules in the CAD system.

Calculation of centrifugal force on each ball:

$$\alpha = 0$$
$$fc = \frac{UBM}{N1}\left(\frac{\Omega}{60}2\pi\right)^2$$

Calculation of ball balancing moment:

$$BMR1 = {UBM1}/{N1}$$

Calculation of wall height:

$$HT1 = \Phi_1 + 0.005$$

Calculation of wall thickness:

$$\sigma_d = {\sigma_y}/{5}$$
$$T1 = \sqrt{{3fc}/{\sigma_d}}$$

Calculation of ball mass:

$$BM = \pi\frac{\Phi_1^3}{6}\rho_B$$
$$\rho_B = 7800$$

Calculation of disc radius:

$$RT1 = \frac{BMR1}{BM} + T1 + \frac{\Phi_1}{2}$$

Tasks

1. Set up a DSM for parameters, intermediate variables and output variables using given input parameters and relations above
2. Set up the CAD model with parameters and variables coupled to the parametric geometry. Define relations based on above equations. Insert print messages in all rules so that execution can be followed and study the sequence of rules firing. Select starting values from the list above and run the system to determine output variables.
3. For standard applications ABS plastic is used with yield strength of 15 MPa. If fc exceeds 175 N, material is changed to PETP, with yield strength of 35 MPa. Insert a rule for this purpose and check the effect of increasing speed or ball diameter.

4. In the first case the allowable thickness of the balancer, and thereby the ball diameters, were given. Assume that instead the outer diameter is given and we want to find the correct ball diameters. We do not want to write a new code, but merely add the required relations to the existing rule base and re-execute. Add the following relations:

$$\Phi_1 = \sqrt[3]{6\frac{\frac{BMR1}{R1}}{\pi\rho_B}}$$
$$R1 = RT1 - T1 - {}^{\Phi_1}/_2$$

These two equations are however coupled and would result in "circular reference". To uncouple them make the approximation

$$R1 = RT1 - T1 - 0.009$$

Deactivate the rule that fires RT1 and set this value manually. Rerun the system and see what rules fire and in what order.

5. A more demanding task: Solve the coupled problem by partioning using an equation solver either internally in the CAD system or by an external solver for:

$$\begin{cases} R1 = RT1 - T1 - {}^{\Phi_1}/_2 \\ \Phi_1 = \sqrt[3]{6\frac{\frac{BMR1}{R1}}{\pi\rho B}} \end{cases}$$

Tutorial 4: A pre-study for a design automation system

This tutorial encourages the reader to become familiar with the procedures discussed when planning a computer system for automatic design.

Select a limited design problem that you are well familiar with technically. Write a short essay, 5–10 pages that relates to a chosen application of design automation. Briefly present this task and discuss:

- Which are the business motives for Product Variety and how is variety achieved?
- What are the motives to automate the design process?
- Which are the categories of knowledge used. Use the categories presented in the book or some other classification. Evaluate applicability and relevance of classification method.
- What human cognition processes are involved.
- From this perspective, is it meaningful to automate the process?
- Is the product technically mature with all required know-how available?
- How well structured is the design problem?

- What computerised solution methods should be applied? Discuss strength and weaknesses.
- How should design knowledge be documented?
- How should knowledge base and processing interact with commercial software?
- How should the system be maintained and what are the security aspects?
- Other vital issues that must be considered early.