DIVISION OF COMPUTER SCIENCE

A Comparison of Development Methods used in Traditional Engineering and Software Engineering

Technical Report No.147

A. Mayes

November 1992

A comparison of development methods used in traditional engineering and software engineering.

Audrey Mayes

November 92

1 Introduction

The term 'software engineering' is said to be contradictory [1] as there is little scientific reasoning behind the processes used. It was felt that the traditional engineering industry might contain some ideas which could be applied to software engineering in order to improve the scientific basis of software engineering. There are obvious differences between traditional engineering products, and software products but sufficient parallels have been identified [2] for a comparison of methods to be considered justified. This report therefore concentrates on identifying similarities and possible 'lessons to be learnt' rather than looking for differences between the two fields. It is written from a software engineer's point of view and takes a general view of both fields.

The following sections show the areas of comparison.

Section 2 The product life cycle

Section 3 The design models produced

Section 4 The planning methods used

Section 5 Reuse of components

Section 6 Conclusions

2 The product life cycle

The traditional engineering design process [3], [4] begins with the identification of a need for a new or improved product. This need is then analysed to produce a requirements specification or product design specification (PDS). The PDS contains details of the objectives to be achieved by the required product. These details cover all aspects from the required performance to constraints such as cost, materials to be used, appearance, life span, reliability, documentation required. A British Standard for specifications, BS PD6112 was introduced in 1967. This standard contains a comprehensive list of items for inclusion in the document. The person responsible for writing a specification has to choose the relevant items for the product. BS PD6112 has a strong bias towards the engineering/manufacturing industry and was superceded in February 1991 by BS 7373 which is applicable to a wider field. The Computing Services Association was represented on the committee, indicating that the standard should be suitable for use in software engineering.

The next stage is the conceptual design or synthesis stage. Many different possible designs are produced to satisfy the requirements. The designs can be produced directly with reference to existing components or by first dividing the required product into subsystems which may contain existing components. All the designs are evaluated. The most significant requirements in the PDS are chosen for the criteria for the initial evaluation. In the case of consumer products, the outside of the product is vitally important and is given high priority, but for something like an oil-rig the function is the higher priority. The design which best meets the criteria is further developed and

evaluated against all the requirements. At the end of this stage the final layout of a design which most closely matches all the criteria is available. The linking between the components and their basic designs will have been established.

After the conceptual design is complete, the final design is developed. This deals with the designs of subsystems and components of which the whole design is composed. In this phase it is necessary to take into account the predefined interfaces between the components. Each subsystem or component is developed within an established context. A specification of each component is produced. This is similar in form to the original product design specification. The same processes of analysis, conceptual design and evaluation are used at this stage but at a detailed level.

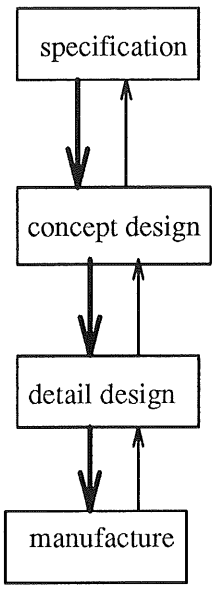
The final stage in the design of the product also concerns ensuring it can be produced efficiently. This may necessitate changes to

1. minimise the cost of the components and their assembly into the final product
2. make the development time needed for the production process as short as possible
3. ensure the product meets its specification.

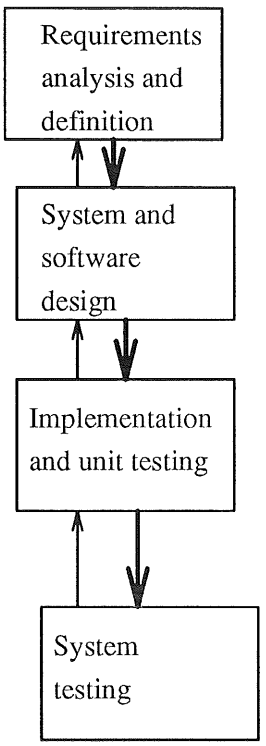
The effects of changes in the design should be considered. It is possible that one implementation will give the same functionality but will compromise a different requirement.

The traditional design process appears to follow the same type of pattern as the 'waterfall model' in software engineering [5]. Figure 1 shows diagrams of the development cycles. The specification phase in the traditional engineering cycle corresponds to the requirements analysis and definition phase in the software cycle which establishes the system's services, constraints and goals. The concept design and detail design corresponds to the system and software design.

Testing in the software industry is seen as a separate activity in the development cycle and consumes a large amount of resource. D'Ippolito [6] states that a chemical production plant was built without any tests of the design being necessary. This was possible because the models of all parts, that is the components, interfaces and controls were validated prior to inclusion. The only testing performed was to ensure that the plant was an accurate implementation of the design. He also points out that engineers are now taught to design the unit operations and the objects that provide them, such as pumps, instead of being taught to design a particular kind of structure such as the chemical plant. He suggests that software engineers can proceed in the same fashion. This view of testing in traditional engineering is not universal. Leech [7] states that it is almost certain that a product or its sub-assemblies will not work when first tested. Many redesigns and re-tests are said to be needed, leading to increased costs and causing difficultites in initial cost estimation. The redesigns and re-tests of parts after initial completion of the product suggest that the sub-assembly, interfaces and control models were not validated before assembly in the final product. However, Pugh [3] indicates that testing is included as part of the specification of each component and is not a separate activity. Software testing proceeds from testing the small units of design or code to ensure that each meets it specification. Integration testing of these parts proceeds to ensure that the complete system performs as required.



Traditional life cycle model, adapted from [3]



Software life cycle model, adapted from [5]

Figure 1: Traditional Engineering and Software Engineering Lifecycle Models

The design of the manufacturing process has not usually been considered until the product designs are complete. This has led to some long delays. A new technique called concurrent engineering is being developed. This involves carrying out as much of the process design as possible simultaneously with product design in order to cut down development time. Other ways of decreasing the development time include using existing equipment if possible, designing parts with a long lead time as early as possible, using new complex parts only if they are vital to competitive edge and making prototypes as soon as possible to prove the functionality of the design.

Product designs can change by evolution. For example a component could be replaced by an improved version. This is initially cheaper than producing new designs, but can lead to compatibility problems. At this stage a new design has to be produced.

Maintenance in traditional engineering consists of taking action to ensure that the product continues to meet its original specification. This includes replacing parts that fail due to faulty design and manufacture as well as replacing worn out parts. The fixing of errors in software engineering equates to the replacement of parts due to faulty manufacture or design. Many software engineers also introduce a second interpretation of maintenance, that is they include adding unforseen functionality to the system. It is this aspect that leads to most of the problems with loss of program structure and readability. The traditional engineering industry do not consider increased functionality to be maintenance. Extra functionality such as measuring pressure and temperature instead of just temperature is considered to be a new product and therefore to require a redesigned product. This product may be based on the previous version but is still considered

a new product.

British Standard 5750 can be used in all engineering disciplines. It deals with Quality systems and attempts to cover the aspects of quality not covered by the technical specifications. Many of its sections contain guidelines to best practice and as such are applicable to any manufacturing or service industry.

2.1 Good designs

Some objectives for good designs of manufactured products have been suggested [7].

1. Simplicity - use the simplest workable design. Reduce the number of parts and make the product as small as is compatible with other requirements.

   Metrics have been developed [3] to calculate the efficiency of a design. One metric is called the complexity factor and takes into account the number of parts(Np), the number of types of parts (Nt) and the number of interfaces (Ni).

   $$complexity\ factor = \sqrt[3]{Np * Nt * Ni}$$

   An alternative metric is the design efficiency which uses the minimum number of parts (NM) and the time taken to assemble them (TM).

   $$design\ efficiency = 3 * NM/TM$$

   The correlation between these two figures is not known but a high design efficiency correlates with a low complexity factor.

2. Lightness - less material used therefore cheaper and it costs less to transport. This might translate to a requirement to have the program take up as little space as possible. This is becoming a less important requirement as hardware costs decrease.

3. Standardisation - use standard parts as much as possible because they have already been proved. The cost of developing a special design for a small industrial motor is estimated to be at least 50% more than the cost of the standard machine. If non standard design is needed it should be compatible with other standard parts. It is apparently assumed that the designer will look in a catalogue to find out the availability of a part which matches his requirements. An underlying support system such as the classification scheme is also assumed.

4. Flexibility - to allow for compatibility with other components and facilitate upgrades. This criterion conflicts with others in that by designing to allow for flexibility there may be a reduction in efficiency eg design to allow for greater stress than expected may lead to added weight.

5. Tolerances - questions such as how fine the finish has to be or how much the diameter of a pipe can vary. These must be realistic to avoid undue cost.

6. Proper use of manpower - designers must be aware of the expertise of the production staff. Any training required must be initiated as soon as possible.

7. Design for reliability - this appears to consist mainly of failure analysis! It also involves designing to avoid accidents and making the occurrence of an accident as safe as possible by designing safe ways for the product to fail. Margins for error are added to the system.

There is no agreed list of good design characteristics in object-oriented software. Wirfs-Brock et al. [8] use simplicity as their empirical measure of design quality. They suggest several criteria on which to base a judgement of the design. A simple design will have:

1. fewer more intelligent classes,
2. more subsystems encapsulating the application specific functionality,
3. few contracts per class
4. deep inheritance heirarchies.

Rumbaugh [9] gives no guidelines for selecting a good design. He suggests that the design should be optimised to make the design more efficient. Optimisations include storing a derived value to save time recalculating every time the value is needed.

Booch [10] suggests evaluating designs by comparing facets such as computational efficiency, synchronisation problems, independence from hardware and simplicity to implement.

It might be possible to devise a metric for comparing object-oriented software design quality. This could relate to the design efficiency rating used in traditional engineering by linking the size of interfaces, the number of classes and some measure of the depth of hierarchies.

A law, called the Law of Demeter, has been developed [11] to help formalise the meaning of 'good style' for object-oriented programs. It is a way of restricting and documenting the dependencies between classes in order to increase the ease of modification of the classes. Adherence to this law is said to help in the production of a 'good design' and could be used to assess designs by checking if they follow the law. Valid violations of the law are said to exist. A law such as this is not a simple design metric such as is available to traditional engineers.

Some design metrics have been suggested for comparing the complexity of structured programs. These suffer from difficulties such as being applicable to only one style of programming or not taking all dependencies into account [5].

3 Design models

In order to allow designs to be effectively communicated, models of the products are developed. In traditional engineering these models can take the form of prototypes, physical models or drawings. Different types of drawings are produced to serve different purposes [12], [13].

Drawing type	**Purpose**
block diagram	relationships between elements in a group. This could be processes or systems.
drawing of principle	uses symbols to show the system eg electrical circuit
layout	relationships between elements in a detailed design
detail drawing	to fully specify an object
assembly drawing	to show the structure for the finished object
exploded views	to show the arrangement of parts in a final design. For describing repairs and spare parts.
overlay drawings	to show the relative positions of inner and outer parts.
bar charts	show dependence of one or more parameter on another.
graphs	relationship between two parameters.

Diagrams are also used to show the control systems showing the sequence of operations. Functional diagrams showing the product working are also used.

Many of the aspects of the products being modelled in these diagrams have no relevance to software. One interesting point is that one type of diagram is used to show the structure of an object for the manufacturer and a different diagram is used for describing spares. Block diagrams, drawings of principle and layouts appear to be used during the early stages of design as the basis for the production of the detailed models used for production and maintenance. This appears to contrast with an object-oriented ideal of adding detail to one model [14].

Graphical standards for engineering drawings were introduced in the 1950's. All diagrams should conform to these standards. The use of standards allows engineers from one branch of engineering to understand diagrams produced by an engineer working in another branch. These standards do not impose a certain style on the designer nor do they limit innovation. The first point is noted by D'Ippolito and Plinta [6]. They highlight the fact that architectural designs can be produced in many styles.

There are many different methodologies used to design software systems, for example SSADM, JSD, and a variety of object-oriented design methods are currently being devised. Each of these methodologies uses its own notations and leads to systems with different design characteristics. The choice of notation tends to determine the system style produced. The use of different notations leads employers to ask for experience of methodologies and programming languages rather than expertise in a field of work.

A standard notation showing all aspects of a system might allow the same analysis and designs to be interpreted in different ways and allow many styles of systems to be evaluated for an application at a late stage in the development rather than being tied to one style because of the chosen methodology. This would also allow designers to specialise in industries rather than in methodologies.

A difficulty with both traditional and software engineering is that the development of a product does not consist of a series of steps carried out sequentially. Iteration is needed to allow for the greater understanding of the system gained as development progresses. This can lead to problems of consistency between the models. The availability of CAD/CAM tools has eased the problem of keeping the information on the various diagrams compatible and consistent in the traditional engineering industry. CASE tools have been developed to support some of the methodologies used in software engineering.

D'Ippolito [6] points out that it must be possible to trace the requirements from the initial documents through the models to the final product. The final product in a traditional engineering environment closely matches the design. Any changes made during production must be recorded to allow the same changes to be made in subsequent products. The software engineer is not under the same constraints because of the ease with which software can be reproduced. It is important that all changes are noted to allow so called maintenance of the systems to be carried out. There are standard documents produced to detail the information required to maintain traditional engineering products. System documentation performs a similar function in software engineering but there is apparently no industry wide standard. A reliable method of extracting design information from code is required to ensure that the final design is properly documented.

4 Design Planning methods

Traditional engineering and software engineering both use the same methods for planning the projects. These include Gantt charts, network planning and Programme Evaluation and Review Technique.

5 Reuse of components

Most components used in both the electronics and manufacturing industries are readily available off the shelf [13],[15]. It is estimated that 80%-90% of a product consists of 'off the shelf' parts. These parts can be used because they are

1. general purpose with a well defined standardised interface, eg. size and/or connections to the outside world,

2. work almost as efficiently as a specially designed part.

The proportion of standard parts varies in the different branches of engineering. Electrical goods contain more standard parts because of the relatively limited variety of those parts. Civil engineering is at the other end of the spectrum with standard processes rather than standard parts being used.

Leech [7] lists the use of standard parts as one of his design objectives. A non standard part is estimated to cost at least 50% more than a standard part.

6 Conclusions

Software engineering and traditional engineering have the same high level objective, in that they are both trying to produce a complex system which will fulfill a need and meet performance criteria.

There are many similarities in the methods used. The basic steps in the cycle are the same for any development project. The need or problem has first to be identified. The next stage is analysis to produce a full list of requirements. The design can then be produced, made and tested.

The design stage of both branches of engineering has some similar characteristics. The required system is divided according to some criteria, such as functional subsystems or components. This division is carried out in order to simplify a complex system. Stovsky and Weide [16] have noted this similarity. They also note that information hiding is used in both traditional and software engineering to emphasise the external characteristics of a component which are important to the user.

There also appear to be differences at the design stage. In traditional engineering there is great emphasis put on producing several different designs which are then evaluated against the required criteria before a final design is produced. In software engineering it is apparently the norm to follow one methodology and produce one design which is then implemented. Any evaluation of the design is usually informal. This suggests that the software branch of engineering undervalues the design stage of development.

A manufacturer of a product is supplied with several different views of the required product. Changes to improve quality and cost can be made at this stage in much the same way as people implementing software systems are free to choose the algorithms used.

Metrics exist to compare the design complexity of manufactured products but not to compare the designs of object-oriented software systems.

Engineers have many sets of models which can be understood by all engineers. Different models are relatively more important in different industries and some diagrams are not needed for all products. For instance overlay diagrams are not needed for the design of a simple table. All the models have defined standards. Some standards could be applied to software engineering. Stovsky [16] has identified the development of standard graphical notations for software designs to be desirable, emphasising that this need not be a single notation. The author's view is that more than one notation would be required to model clearly all aspects of a complex system. Several models of a system could be produced to show different aspects of the system. The implementation could then be based on the view which gives the 'best' system according to the required criteria as defined by the requirements specification. The object-oriented design method suggested by Rumbaugh [9] seems to go some way towards this.

A major difference between traditional engineering design and software design is the availability of standards for the production of models in traditional engineering. Information is then represented in a standard form which allows the effective transfer of knowledge. These standard models can be applied in all branches of traditional engineering and do not limit the designer to one style nor prevent innovations. The software engineering approach is to follow a particular methodology and produce the models associated with that methodology. The use of a methodology has a strong influence on the designs produced and limits the ability of software engineers to understand the models designed by unfamiliar methods.

References

[1] B.J. Cox. There IS a Silver Bullet. *IEEE Software*, 9 1990.

[2] J. A. Mayes and C. Britton. Are there any parallels between object-oriented system development and other branches of engineering? TR 139, Hatfield Polytechnic, 1992.

[3] S. Pugh. *Total Design.* Addison-Wesley Publishers Ltd., Wokingham, England, 1990.

[4] B Hawkes and R. Abinett. *The Engineering Design Process.* Longman Scientific and Technical, Harlow, Essex, 1984.

[5] I. Sommerville. *Software Engineering.* Addison Wesley, third edition, 1989.

[6] R. D'Ippolito and C.Plinta. Software Development Using Models. In *Proceedings 5th International Workshop on software Specification and design*, 1989.

[7] D. J. Leech and B.T. Turner. *Engineering Design for Profit.* Ellis Horwood, 1985.

[8] R. Wirfs-Brock, B. Wilkerson, and L. Weiner. *Designing Object-Oriented Software.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented Modelling and Design.* Prentice-Hall International Editions, Englewood Cliffs, New Jersey, 1991.

[10] G. Booch. *Object Oriented Design with Applications.* Benjamin/Cummings Publishing Company, Redwood City,California, 1991.

[11] K.J. Leiberherr and I.M. Holland. Assuring Good Style for Object-Oriented Programs. *IEEE Software*, 9 1989.

[12] E.Tjalve, M.M. Andreasen, and F. Frackmann Schmidt. *Engineering Graphic Modelling.* Newnes-Butterworths, London, 1979.

[13] Pat Hamilton University of Hertfordshire. private communication.

[14] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, Inc, Englewood Cliffs, New Jersey, second edition, 1991.

[15] Simon Trainis University of Hertfordshire. private communication.

[16] M. Stovsky and B. Weide. The Role of Traditional Engineering Design Techniques in Software Engineering. In *SEKE Proceedings 2nd International Conference on Software Engineering and Knowledge Engineering*, 1990.