Module 1– The Role of Software Design

Dr.N.Bala Ganesh,
AP/ SCORE/ VIT University

The Nature of the Design Process

Introduction to Design

- We ride in cars, trains, aeroplanes; we live in houses or flats; we use everyday domestic appliances such as washing-machines, television sets, vacuum cleaners; we sit on chairs, lie on beds, walk around in shoes; we play games, listen to music.

- All of these are artifacts because they have been devised and created by human beings, and **all of them in some way are the products of some form of design process** – whether a good one (shoes are comfortable, a washing machine works reliably) or a poor one (the flat roof of a house leaks, or the chair collapses when the user leans back on two legs).

Introduction to Design

- **No-one is likely to deny the importance of using well-proven design practices** for the design of motorway bridges, aeroplanes and buildings, not least because of the safety issues concerned with the use of such objects.

- **Yet equally, good design is important for a wide range of less safety-critical objects** – such as a domestic refrigerator: we do not want to have to de-ice it continually, nor to replace bottles that fall out when we open the door.

- Similarly, we also want to have well-designed footwear so that we do not find ourselves suffering from foot complaints

Introduction to Design

- Design is just as important with software systems also.
- **Most people will readily accept that the software used in an aeroplane needs to be well designed and rigorously tested**, not least because they might find themselves as passengers on that aircraft one day.
- **Yet good design is equally desirable for smaller systems too,** since the user still requires efficiency and reliability

Introduction to Design

- A word processor may not be a safety-critical item, but its user is unlikely to appreciate the occasional lost paragraph occurring at apparently random points in a document.

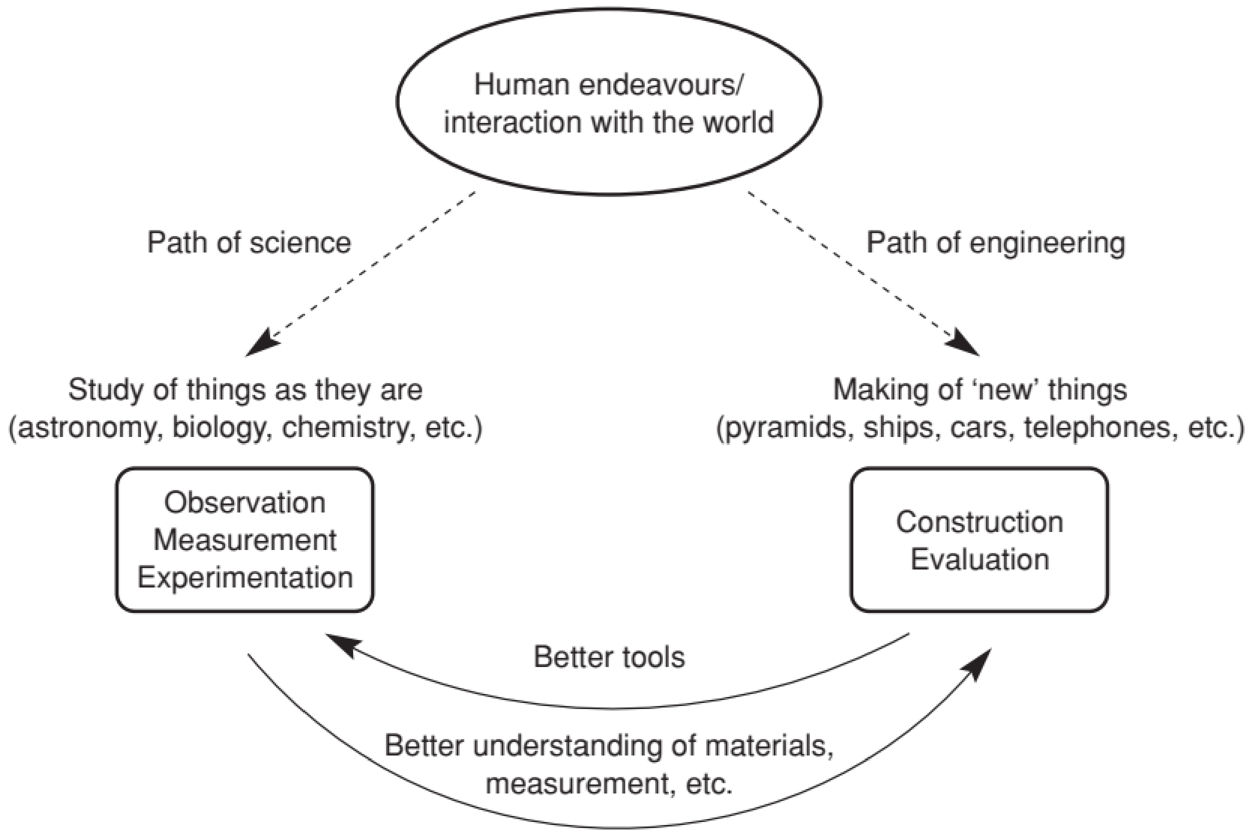
- **It may not be appropriate or cost-effective to employ the same techniques for designing a word processor as for designing safety-critical avionics systems**, but the need for a well-designed product is still there.

Introduction to Design

- The following concise description of the **design process** is more than sufficient to show that its form **is very different from that of the scientific approach to problem-solving.**

- *'The fundamental problem is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct. The final outcome of designing has to be assumed before the means of achieving it can be explored: the designers have to work backwards in time from an assumed effect upon the world to the beginning of a chain of events that will bring the effect about'*

Introduction to Design


Human endeavours/ interaction with the world
Path of science
Path of engineering
Study of things as they are (astronomy, biology, chemistry, etc.)
Making of 'new' things (pyramids, ships, cars, telephones, etc.)
Observation Measurement Experimentation
Construction Evaluation
Better tools
Better understanding of materials, measurement, etc.

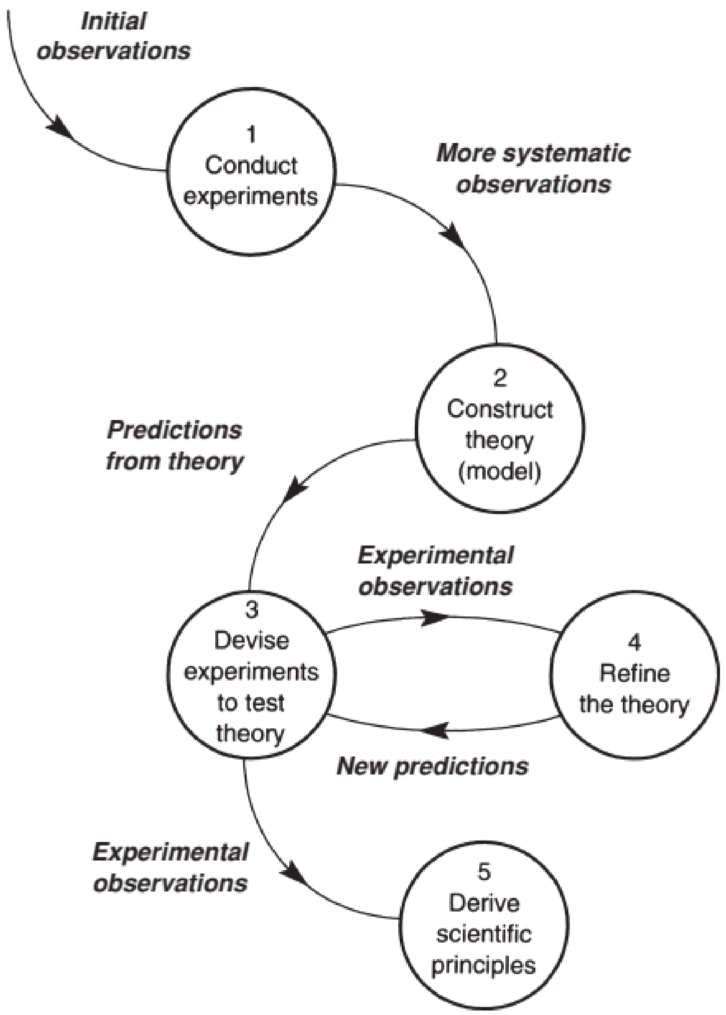
The complementary nature of scientific and engineering activities.

Introduction to Design

- Of course, these two paths are not isolated from one another. Indeed, the interplay between them has always provided an important element for both.

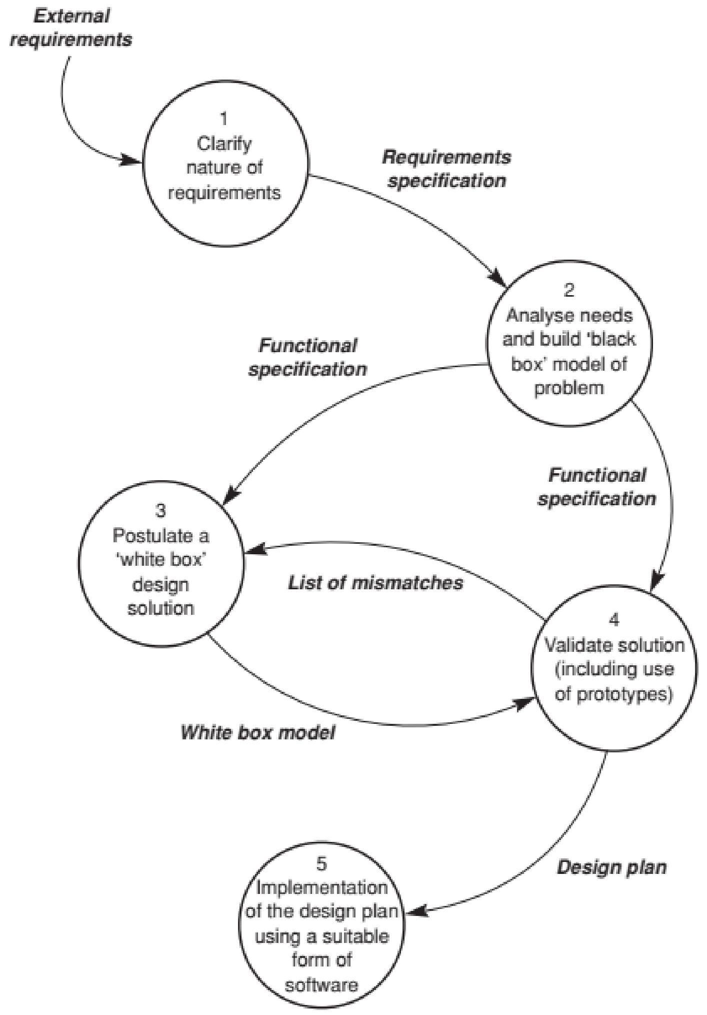
- The **products of ‘engineering’ have formed important tools for advancing scientific knowledge** (from clocks to cyclotrons), **while the observations and measurements of science have provided the understanding of their basic materials that engineers need in order to use them** in new ways and to most effect.

Introduction to Design


Initial observations
1 Conduct experiments
More systematic observations
2 Construct theory (model)
Predictions from theory
Experimental observations
3 Devise experiments to test theory
4 Refine the theory
New predictions
Experimental observations
5 Derive scientific principles

The nature of scientific analysis.

Introduction to Design



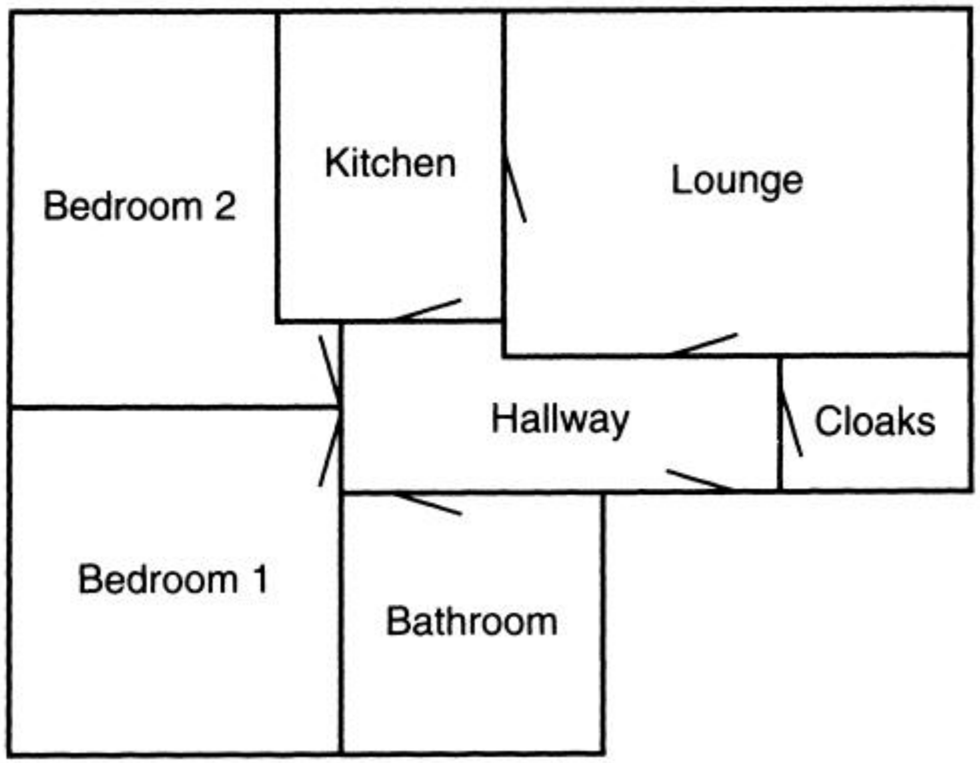
External requirements
1 Clarify nature of requirements
Requirements specification
2 Analyse needs and build 'black box' model of problem
Functional specification
Functional specification
3 Postulate a 'white box' design solution
List of mismatches
4 Validate solution (including use of prototypes)
White box model
Design plan
5 Implementation of the design plan using a suitable form of software

A model of the design process.

1. **Propose a solution;**
2. **Build a model of the solution;**
3. **Evaluate the model against the original requirement;**
4. **Elaborate the model to produce a detailed specification of the solution.**

Introduction to Design

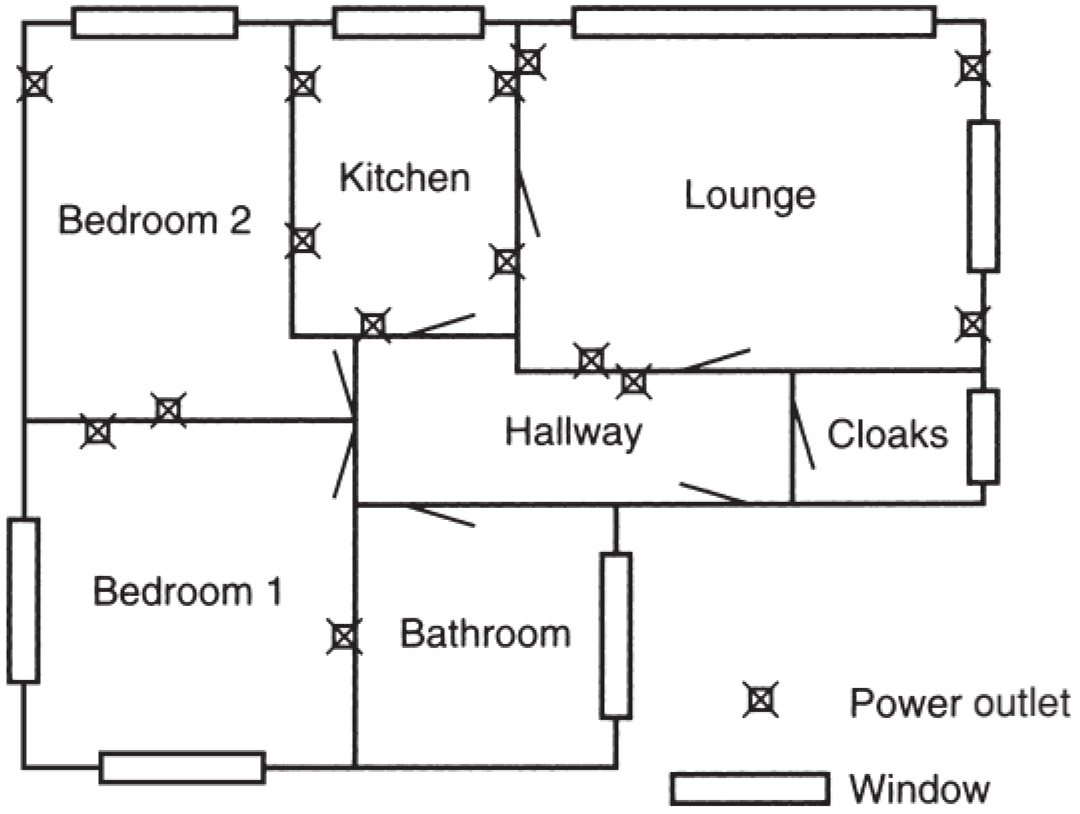
CASE STUDY 1: Moving house (Positioning furniture in new House)



Outline plan of a single-storey house.

Introduction to Design

CASE STUDY 1: Moving house (Positioning furniture in new House)



Expanded plan of a single-storey house.

Introduction to Design

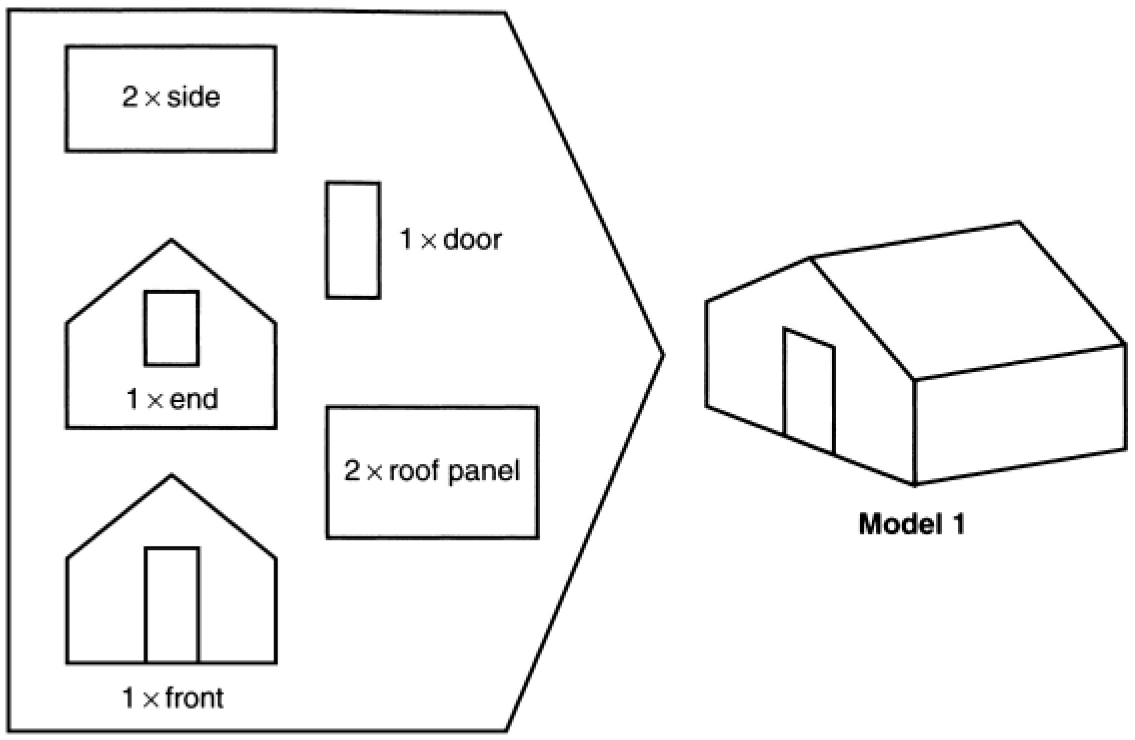
- From this relatively simple case study, we should be able to see some of the ways in which the process of design differs from that of scientific method.

- Rather than being able to identify a ‘right’ coordinate system that allows us to separate the variables and solve for each separately, we have to build a relatively complex model, and then make adjustments to this in order to produce a solution.

- There may be many possible ‘solutions’, and we may well have no very strong criteria that can be used to help us in choosing from a number of the better ones.

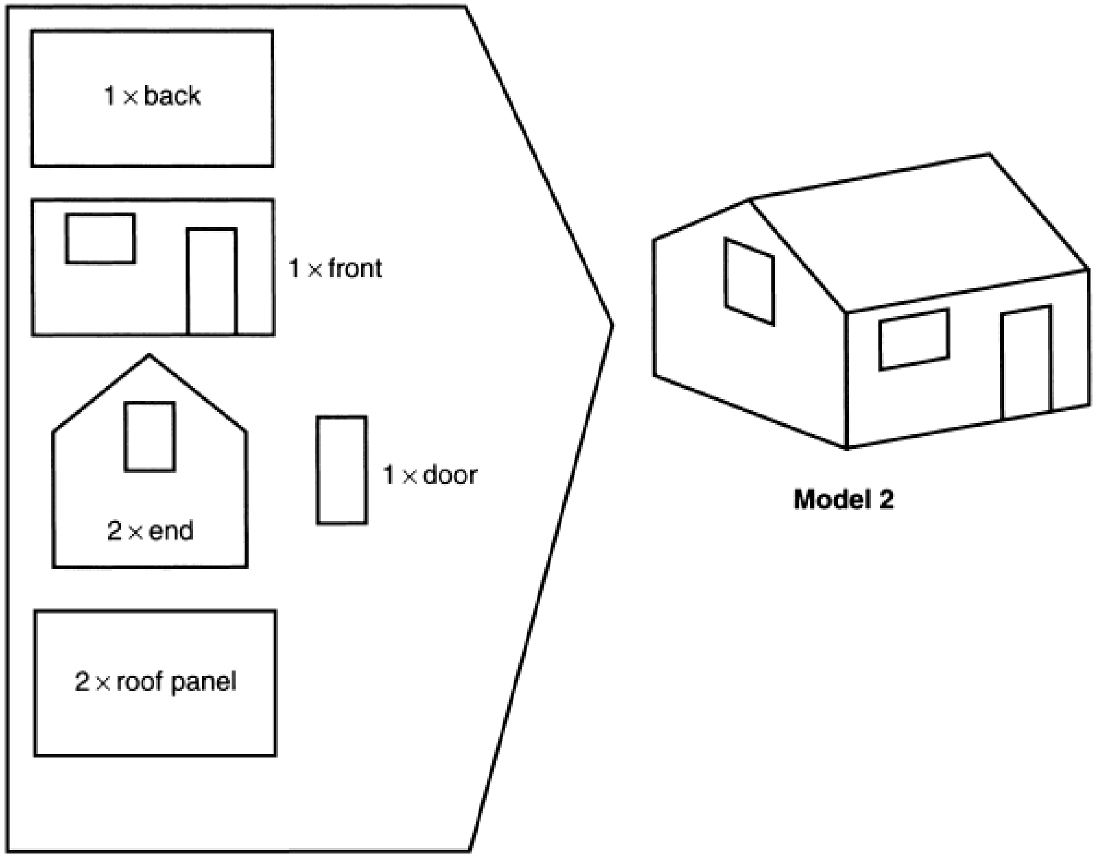
Introduction to Design

CASE STUDY 2 – Garden Shed



Introduction to Design

CASE STUDY 2 – Garden Shed



Introduction to Design

- A shed should be assembled from a set of prefabricated panels for the sides and roof. Assuming that the boarding on the sides runs horizontally, each panel should be of a height that allows it to be constructed from a whole number of boards.

- **Where possible, the panels used in one shed should be of a size that allows them to be used in another (Reusability)**: for example, the side panel for a small shed could also be used as the back panel for a larger model.

- These effectively form constraints upon the design process too, and the **designer will also need to consider a number of practical issues about size and shape of the range of products**

Introduction to Design

- Given the original requirements that were identified by the company, and the constraints created by the nature of the material and the need for reuse of components, the outcome of the process will be a set of drawings showing the dimensions of each panel, the way that it should be assembled from the basic units of timber, and the ways that the panels can be assembled to create different sheds.

Introduction to Design

- **In one case the end product is an organizational plan for keeping furniture, in the other it is a set of plans and drawings that will be used by the joiners who produce the sheds.**

- For both cases there are constraints operating upon the process, although these have very different forms, and in both cases the designer has some concept of quality to help with making choices.

- In the one case this may be the balance of style and colour for the furniture in a room, in the other it will be a matter of producing a shed that has the 'right' proportions and which is easily fabricated.

Role of Design Activity

- **The principal task for the designer is to specify the best solution to a problem and produce a description of how this is to be organized**. This description then forms a 'plan' that can be used by the eventual implementors of the system.

- **The designers of pyramids and cathedrals will almost certainly have exchanged ideas with, and learned from, their peers**. However, each of the resulting artifacts was a single unique creation, and only with the emergence of engineering practices were there significant new inputs to the designer's learning processes.

Role of Design Activity

- **The principal task for the designer is to specify the best solution to a problem and produce a description of how this is to be organized.** This description then forms a 'plan' that can be used by the eventual implementors of the system.

- **The designers of pyramids and cathedrals will almost certainly have exchanged ideas with, and learned from, their peers.** However, each of the resulting artifacts was a single unique creation, and only with the emergence of engineering practices were there significant new inputs to the designer's learning processes.

Role of Design Activity

- Two of below are worth noting here, not least because of their longer-term implications for software design.

1. **The first of these was the knowledge gained from scientific research.**
   a) As the properties of materials became better understood, and the means of predicting their behaviour became more refined and dependable, so the designer could employ this knowledge in helping to create new structures.
   b) A good example is that of bridge building in the nineteenth century, when engineers such as I K Brunel were able to create structures that were radically different from their predecessors, while also proving to be remarkably durable.

Role of Design Activity

- Two of below are worth noting here, not least because of their longer-term implications for software design.

2. **The second was the concept of reuse.**

a) One consequence of the industrial revolution was the idea of ‘standardizing’ components

b) The existence of such components extends a designer’s catalog – on the one hand offering speedier development of an idea (‘improved time-to-market’) with potentially lower costs, but also introducing new constraints and trade-offs into the design process

Role of Design Activity

- In Two case studies, we can see that the objectives for these are somewhat different in form.

1. **A plan that informs the removal men where each major item of furniture is to be positioned;**

2. **A set of plans that inform the joiner about the dimensions of the panels and how the panels are to be jointed and so on**

- The first of these is largely approximate

- **The second requires a greater degree of precision,** and the degree of tolerance that is acceptable in the product may well need to be specified as a part of the plan

Role of Design Activity

- In Two case studies, we can see that the objectives for these are somewhat different in form.

1. **A plan that informs the removal men where each major item of furniture is to be positioned;**

2. **A set of plans that inform the joiner about the dimensions of the panels and how the panels are to be jointed and so on**

- **The first of these is largely approximate**

- **The second requires a greater degree of precision**, and the degree of tolerance that is acceptable in the product may well need to be specified as a part of the plan

Role of Design Activity

In Software Development,

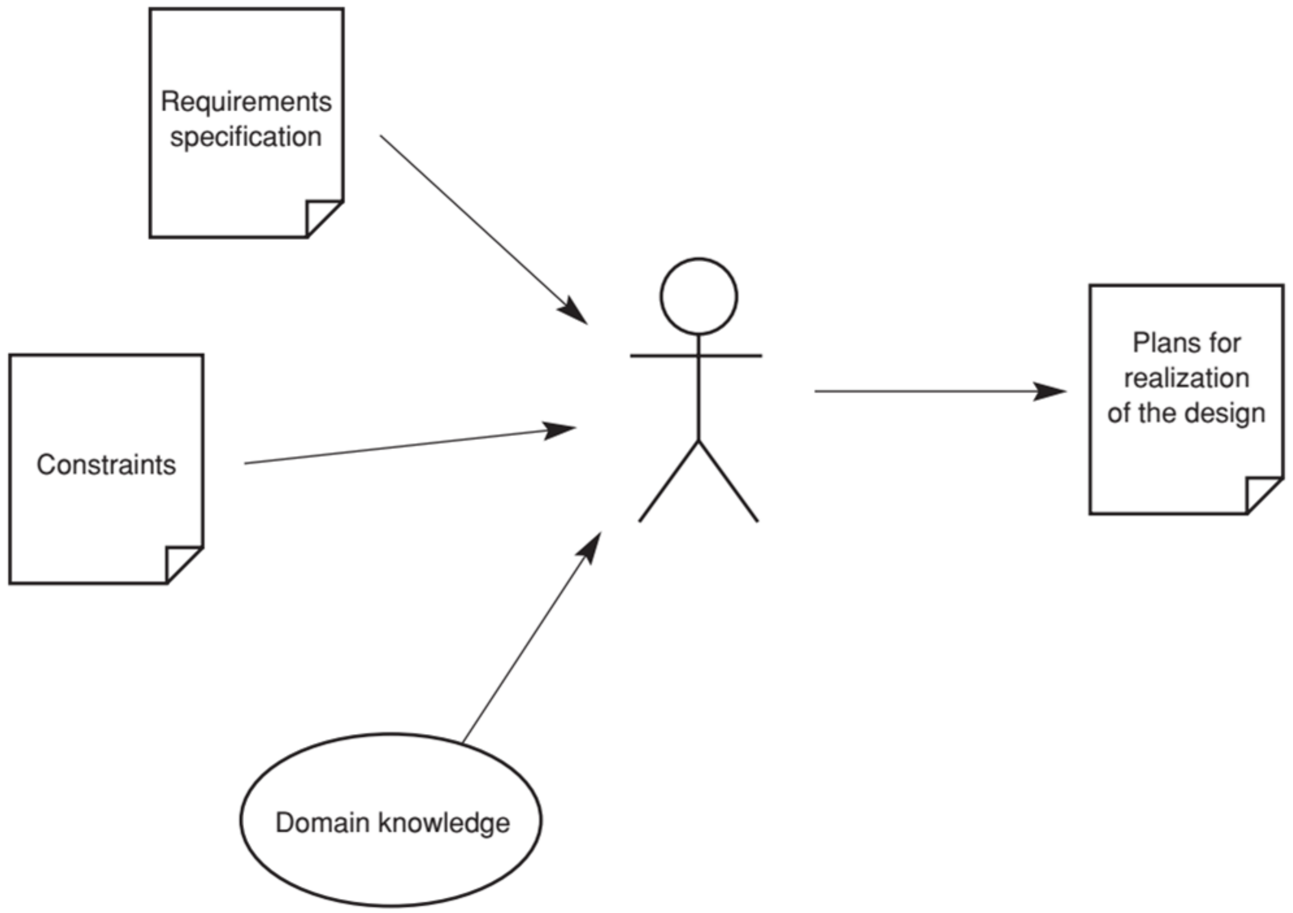
- The main task of the design phase is to produce the plans necessary for software production to proceed.
- The form and extent of the plans will be determined by the design practices and means of implementation chosen, as well as by the size of the system being developed.
- Clearly, in a large project employing many programmers the design plans will need to capture a much wider range of factors than will be needed by the one-person project, where the designer may well be the programmer too

Role of Design Activity

Such software design plans will be concerned with describing:

- The static structure of the system, including any subprograms to be used and their hierarchy;
- any data objects to be used in the system;
- the algorithms to be used;
- the packaging of the system, in terms of how components are grouped in compilation units
- interactions between components, including the form these should take, and the nature of any causal links.

Role of Design Activity


Requirements specification
Constraints
Domain knowledge
Plans for realization of the design

The designer's channels of communication.

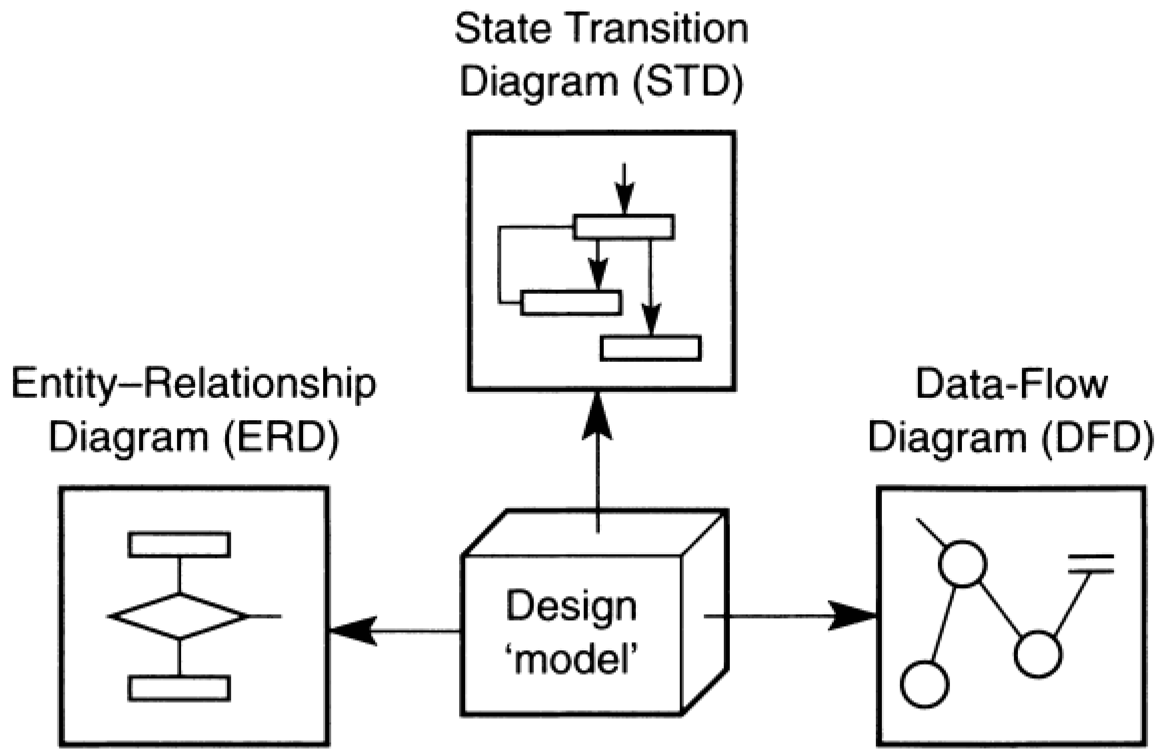
Role of Design Activity

The designer will usually make use of a number of different forms of design representation to help with constructing the model, each representation providing a different viewpoint on the form of a design,

- The more complex the system, the more we need a full set of viewpoints to understand its behaviour, and

- to provide a specification that is sufficiently complete for it to be used as an aid in the construction of the system.

Role of Design Activity


State Transition Diagram (STD)
Entity–Relationship Diagram (ERD)
Data-Flow Diagram (DFD)
Design 'model'

Examples of design viewpoints.

Design as a Problem Solving Process

- **The purpose of design is simply to produce a solution to a problem.**
- The problem will typically be summarized by means of some form of requirements specification, and it is the designer’s task to provide a description of how that requirement is to be met.
- **Design is therefore essentially a problem-solving task.**

Design as a Problem Solving Process

- **The designer has a number of tools to help with the task of problem-solving.**

- **Design methods and patterns** can provide strategies that will help to determine which choice may be most appropriate in a given situation.

- **Representations** can also help with the process of building models of the intended system and with evaluating its behaviour.

Design as a Problem Solving Process

- In combination with these, **abstraction** plays a very important part, since to build manageable models of large and complex systems we need ways of abstracting their critical features for use in forming our models.

- In case study, the outline plan on **squared paper provided an abstraction of the idea of a house**. Clearly a house is a highly complex object, and the two-dimensional plan is the particular abstraction. **For the task of rewiring the house, a quite different abstraction would be needed**, based on the use of some form of circuit diagram

Design as a Problem Solving Process

- **Programmers** are accustomed to working with a wonderfully flexible medium, and so it is all too **easy to be over-influenced by relatively detailed features of programming language structures when thinking about design.**

- The **designer needs to learn to think about a system in an abstract way** – in terms of events, entities, objects, or whatever other key items are appropriate – **and to leave questions of detail, such as the specific forms of loop construct to be used in a particular algorithm, until a relatively late stage in the design process.**

Design as a Problem Solving Process

- Design methods are intended to encourage the designer to think about a system in an abstract way.

- **The effective use of abstraction is a key skill that any designer needs to learn and practice.**

Design as a “Wicked” Problem

- **Design process has may well be a number of acceptable solutions to any given problem.**

- Because of this, the process of design will **rarely be ‘convergent’**, in the sense of being able to direct the designer **towards a single preferred solution.**

- The notion of the **‘wicked’ problem**, which is sometimes used to **describe a process such as that of design, suggests that it is potentially unstable.**

Design as a "Wicked" Problem

Distinguishing properties of wicked problems

1. **There is no definitive formulation of a wicked problem**
   - ➢ The difficulties of specifying the needs of software-based systems are well known, and the tasks of specification and design are often difficult to separate clearly
2. **Wicked problems have no stopping rule**
   - ➢ Lack of any quality measures that can be used to establish that any one system design is the 'best' one possible
3. **Solutions to wicked problems are not true or false, but good or bad**
   - ➢ For many scientific and classical engineering problems, we may be able to identify whether a solution is correct or false.
   - ➢ Software designs usually come in 'shades of grey', in that there are usually no right or wrong solutions or structures

Design as a “Wicked” Problem

Distinguishing properties of wicked problems

4. There is no immediate and no ultimate test of a solution to a wicked problem
5. Every solution to a wicked problem is a ‘one-shot operation’, because there is no opportunity to learn by trial-and-error, every attempt counts significantly
6. Wicked problems do not have an enumerable set of potential solutions

Design as a "Wicked" Problem

Distinguishing properties of wicked problems

7. Every wicked problem is essentially unique
    - So design activities cannot be automated.
8. Every wicked problem can be considered to be a symptom of another problem
    - Writing a computer program, a choice of data structure that helps with resolving one problem may in turn be the source of an entirely new difficulty later.

- From the initial model, the designer makes various choices, and the consequences of any of these choices may well be such as to make further choices much more complicated to resolve. In the extreme, the overall design itself may be shown to be inadequate.

Software Design Process

What is Software

- **Software evolved from Machine level code to Web based coding**

- The way of achieving faster delivery of systems to their users is to employ a higher level of abstraction in terms of what we think of as ‘software’.

- In the 1970s such a step was achieved by moving from writing programs using machine-specific assembler code to writing them in ‘**high-order languages**’.

What is Software

- This freed them from dependence on one particular make and type of computer and also enabled faster production of code.

- An approach offering the potential for a similar step at the design stage is to place greater emphasis on manipulating existing ‘objects’ and ‘components’ rather than designing new ones (**Object Oriented Programming**)

Building Models

The following properties of software as major factors affecting its development.

1. **Complexity** – In software, no two parts are alike and a system may possess many states during execution

2. **Conformity** – Software is expected to conform to the standards imposed by other components, such as hardware, or by external bodies, or by existing software

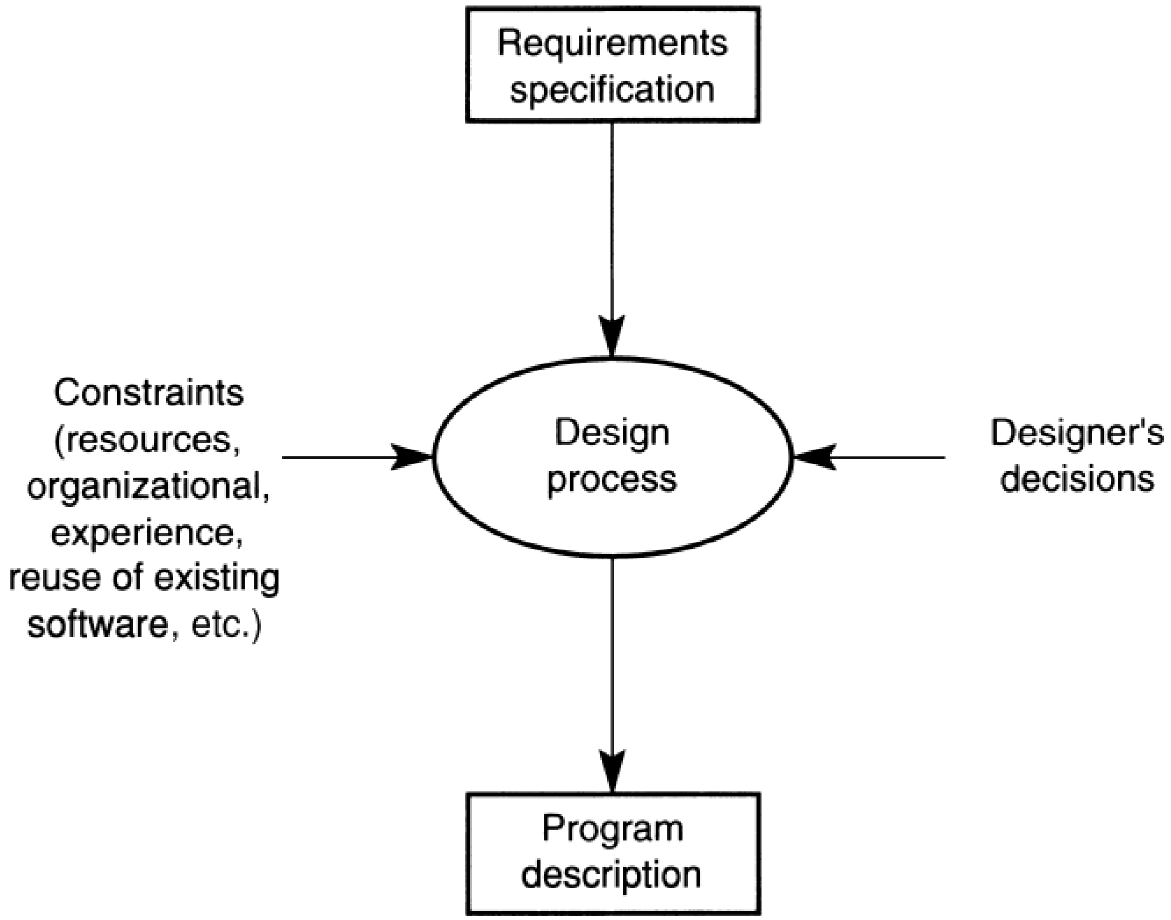
Building Models

The following properties of software as major factors affecting its development.

3. **Changeability** – Software suffers constant need for change, partly because of the apparent ease of making changes

4. **Invisibility** – Because software is ‘invisible’, any forms of representation that are used to describe it will lack any form of visual link that can provide an easily grasped relationship between the representation and the system

Building Models


Requirements specification
Constraints (resources, organizational, experience, reuse of existing software, etc.)
Design process
Designer's decisions
Program description

A general model of the software design process.

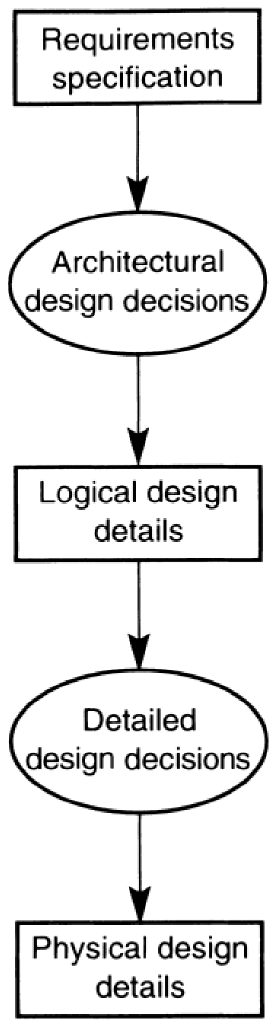
Building Models

In practice, and especially when developing larger systems, the process of design is to be divided into two distinct phases,

1. **In the first phase**, the designer develops a highly abstract model of a solution (the 'architectural' or 'logical' design) in which only the external properties of the model elements are included. (initial black-box partitioning of the problem)

2. **In the second phase**, the abstract 'chunks' of the problem that were identified in the first phase are mapped on to technologically-based units (the 'detailed' or 'physical' design), hence turning the black box into a white box.

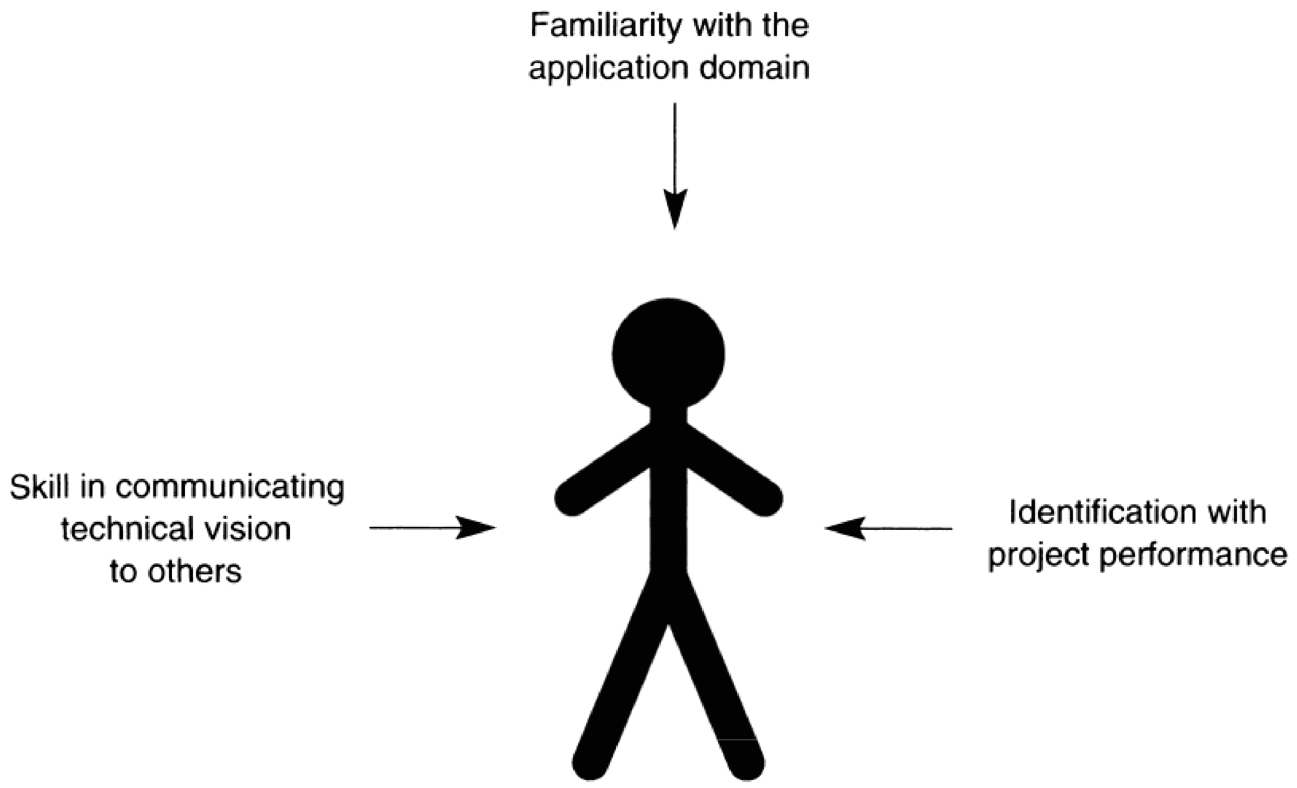
Building Models


Requirements specification
Architectural design decisions
Logical design details
Detailed design decisions
Physical design details

The major phases of the software design process.

Transferring design knowledge

Exceptional designers were observed to possess three significant characteristics.


Familiarity with the application domain
Skill in communicating technical vision to others
Identification with project performance

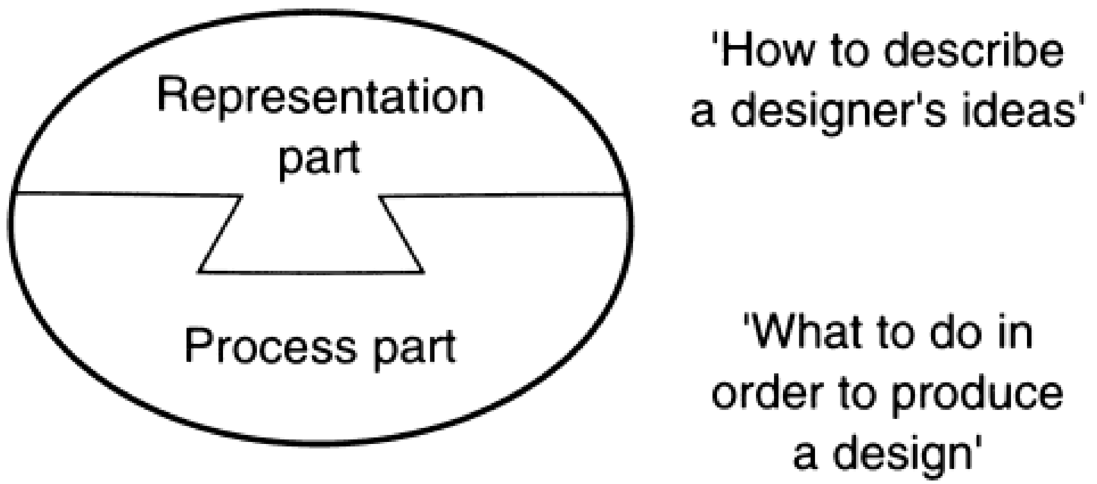
The characteristics of an exceptional designer.

Transferring design knowledge

For our purposes, a software design method can be considered as providing a supportive framework that consists of two major components

1. The **representation part** provides a set of descriptive forms that the designer can use for building both black box and white box models of the problem and their ideas for its solution, and for describing the structural features of the solution to the eventual implementors.
2. The **process part** is concerned with describing how the necessary transformations between the representation forms are to be organized, as well as with any elaboration of their detail that might be required.

Transferring design knowledge


Representation part
'How to describe a designer's ideas'
Process part
'What to do in order to produce a design'

The two major components of a software design method.

Constraints upon the design process and product

1. In case of software design, **we can readily identify many of the constraints imposed upon the form of the product.**

2. **Some may be determined by the run-time environment** (organization of file structures; the 'look and feel' of the user interface), while **others may relate to the need to conform to the conventions required by a chosen architectural style**

3. It is also intended that the eventual system will be constructed by **reusing existing software components**, this **may lead to further constraints** upon the 'acceptable' architectural styles

Constraints upon the design process and product

4. One of the most **important constraints** on the design task and the form of the design itself is that of the **eventual form of implementation (Hardware, OS, Language)**

5. The choice of the programming language is still more likely to be determined by external factors such as **programmer skills and knowledge than by the features of the problem itself**

6. **Many constraints will be identified in the initial specification documents**, although this will not necessarily be true for those that are related to any expectations of reuse.

Recording Design Decisions

- The need to record the decisions of the designer is important, from the viewpoint of the design task itself, and even more so from the viewpoint of the maintenance team who may later need to extend and modify the design.
- Unfortunately, while designers may carefully record the actual decisions, **it is much rarer to record the rationale that forms the basis for the final choice**
- **The recording of rationale is likely to be encouraged if the design process includes any form of design audit**

Designing with Others

- Given the nature of the design process, **it is hardly surprising that designing as a team adds further complications**, many of them in the form of added constraints upon the process, and hence upon the form of the product.

- Many of the very successful and ‘exciting’ software systems have been the work of one or a few ‘great designers’. E.g. Unix, Pascal, Modula, SmallTalk, MS-DOS, Linux and Windows X

Designing with Others

- In the absence of any great designers, designing a system through the use of a team brings **two major additional issues** They are:

1. **How to split the design task among the team, and to determine the interfaces between the parts**;

2. **How to integrate the individual contributions to the design, which may well involve a process of negotiation between the members of the team**

Designing with Others

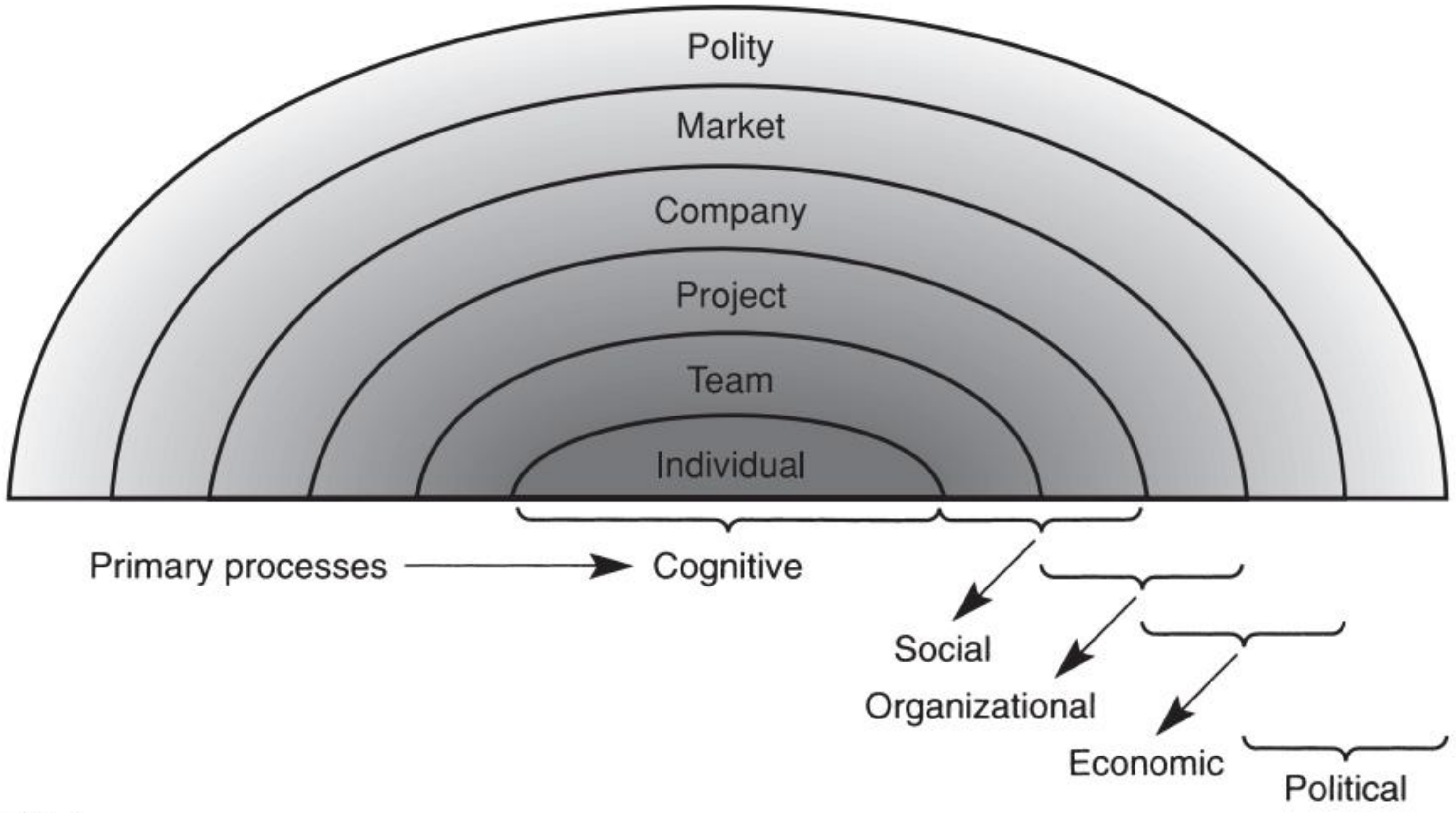
- Researchers have also sought to understand and model the psychological factors that influence team behaviour in designing. The factors discussed in these references include:

1. **The size of a team** (there seem to be pointers that a size of 10-12 members is probably an upper limit for productive working);

2. The large impact that may be employed by a **small subset of the members of the team who possess superior application domain knowledge**;

3. The **influence of organizational issues within a company** (and particularly the need to maintain a bridge between the developers and the customer).

Designing with Others



Factors influencing the software design process.

Design in the Software Development Process

A context for Design

- **There are many ways in which a ‘large software system’ can be developed to meet particular needs or perceived markets.**

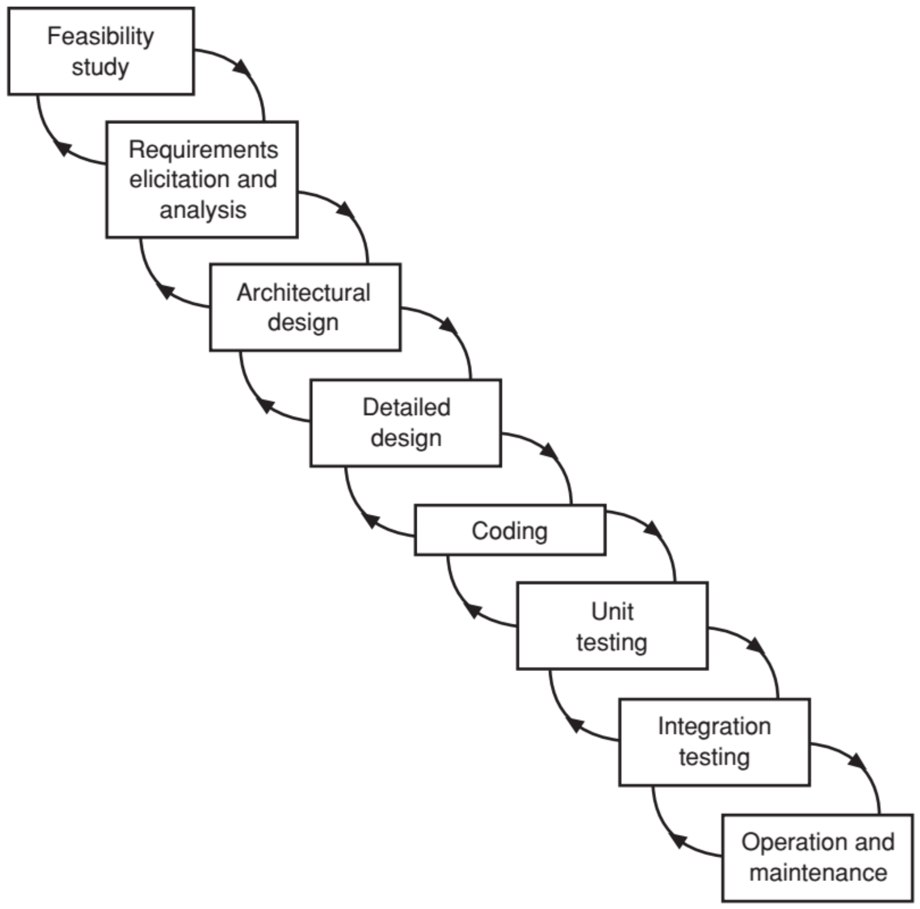
- While each of these generally involves performing largely the same set of activities, the emphasis put upon each one, and the ways in which they are ordered and interwoven, will be very different.

- The **following Table provides a brief overview of some of the forms of development that are commonly adopted**, the reasons why a particular form might be used, and some examples of the domains where its use might be most appropriate.

A context for Design

Development process	Reasons for adoption	Typical domains
Linear (e.g. ‘waterfall’)	Where the needs are reasonably well determined and a ‘longer’ delivery time is acceptable.	Many ‘made to measure’ systems are developed in this way. This approach is one that has been used very widely and is still appropriate for systems that need to demonstrate any form of high integrity.
Incremental	Where there may be a need to demonstrate feasibility of an idea; establish a market position rapidly; or explore whether a market might exist. Providing a rapid time to market is often an important factor for adopting such an approach.	Widely used for ‘shrink wrap’ software of all sorts, such as office packages, games, operating systems. Also, where there is a need to interact closely with the end-users during development.
Reactive	Where evolution of a system is largely in response to what the developers perceive as being needed.	Open source software of all kinds is generally developed in this way, as are many websites.

A context for Design



The waterfall model of the software life-cycle.

Linear development processes

- However formulated, most development processes tend to be implied in terms of the phases identified in the original waterfall structure (shown in previous Figure), and one might well argue that these encompass a set of tasks that always need to be addressed in some way, **regardless of their sequencing**.

- The task classified in the study as ‘**detailed design**’ was particularly stable, **taking around 20 per cent of total effort and 20 per cent of total time**.

Incremental development processes

- A major limitation of a linear development process is that the need to identify exactly what is required of the eventual system has to be addressed right from the start. **In some cases this is effectively impractical.**

- In other engineering domains, such a situation is often resolved by the construction of some form of **prototype** which can be used to investigate both problem and solution.

Incremental development processes

- Forming of a prototype might be different for the case of software, the **basic reasons for building prototypes remain much the same as in other forms of engineering: prototypes are constructed in order to explore an idea more completely than would be possible by other means**

- A useful categorization of the different roles that prototyping can play in software development has been produced by Floyd
  1. **Evolutionary – 'incremental development' of a solution.** The software for a system is adapted gradually, by changing the requirements step by step as these become clearer with use, and changing the system to fit these. In this form, **prototyping is used to develop a product and the prototype gradually evolves into the end product**

Incremental development processes

- A useful categorization of the different roles that prototyping can play in software development has been produced by Floyd

1. **Evolutionary**
   1. One benefit of this strategy is that it may be possible to issue a release of the product while it is still technically incomplete, although usable. (Of course, many software products such as word processors, operating systems, etc)
   2. Later releases can then provide the users with a gradual increase in product functionality
2. **Experimental**
   1. This role is distinguished by **the use of the prototype for evaluating a possible solution to a problem, by developing it in advance of large-scale implementation**. The reasons for doing this may be manifold, including the assessment of performance and resource needs, evaluation of the effectiveness of a particular form of user interface, assessment of an algorithm and so on.
   2. **This form of prototype is essentially intended to be a 'throw-away' item**, and might well be implemented in a form quite different to that which will be used for the final system itself

Incremental development processes

- A useful categorization of the different roles that prototyping can play in software development has been produced by Floyd

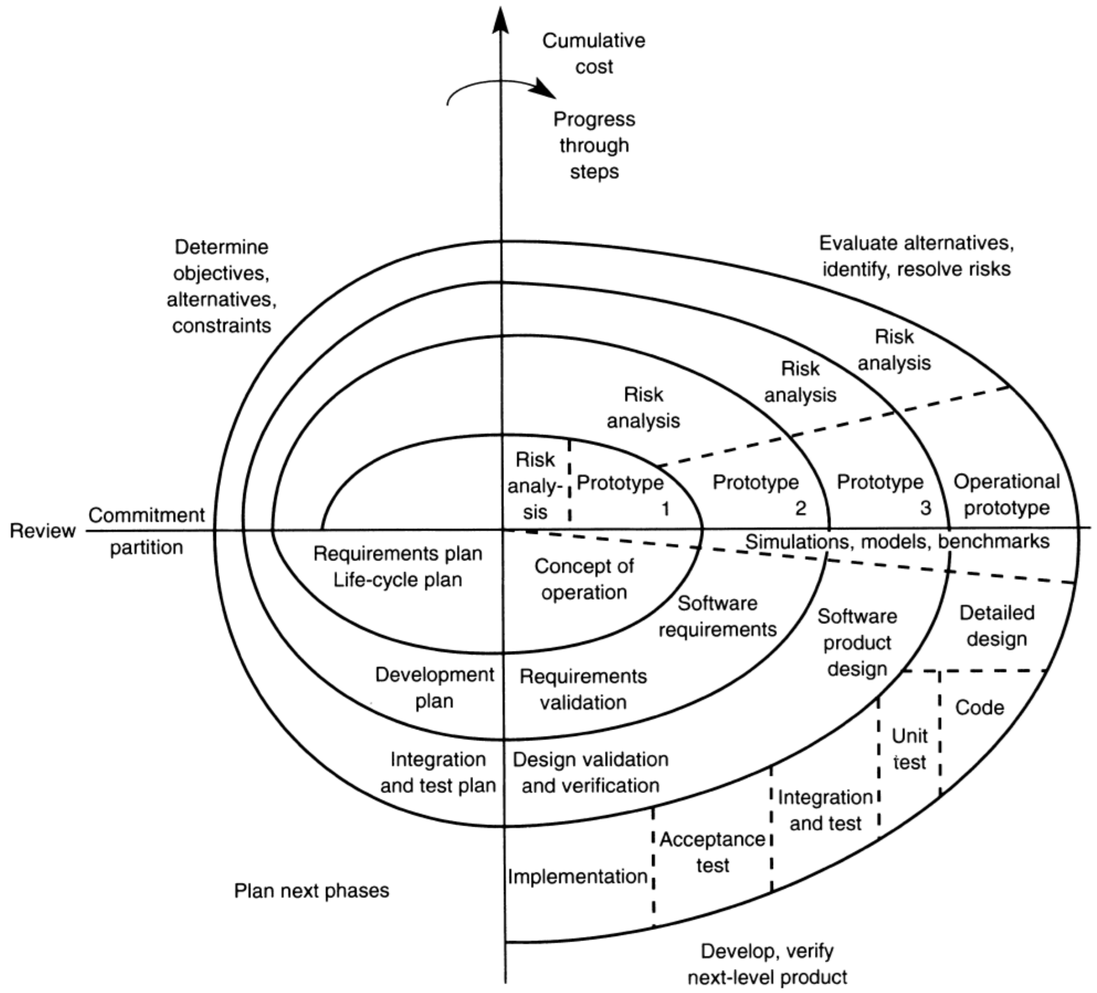
3. **Exploratory**
    1. Here, the role a prototype is used to help with clarifying user requirements and possibly with identifying **how the introduction of the system might lead to the need to modify the wider work practices of an organization**.

    2. **One purpose might be to help with developing an analysis of users' needs by providing a set of possible models for them to use and assess.** Essentially this form of prototype is also likely to be a **'throw-away' item**, and **it can be considered as enhancing the information that is provided from the requirements analysis and the functional specification activities**

Incremental development processes

- **Boehm's spiral model**, while in many ways a rather 'visionary' one when published, is probably the most effective approach to combining experience in using the waterfall model with greater knowledge of how software projects have actually been developed
- **At each stage of the spiral the following activities should occur**:
  1. The objectives of the stage are identified;
  2. The options and constraints are listed and explored;
  3. The risks involved in choosing between options are evaluated;
  4. A plan for how to proceed to the next stage is determined, which may require the development of a prototype, or may more closely approximate to a traditional waterfall step, according to the conclusions of the risk analysis.

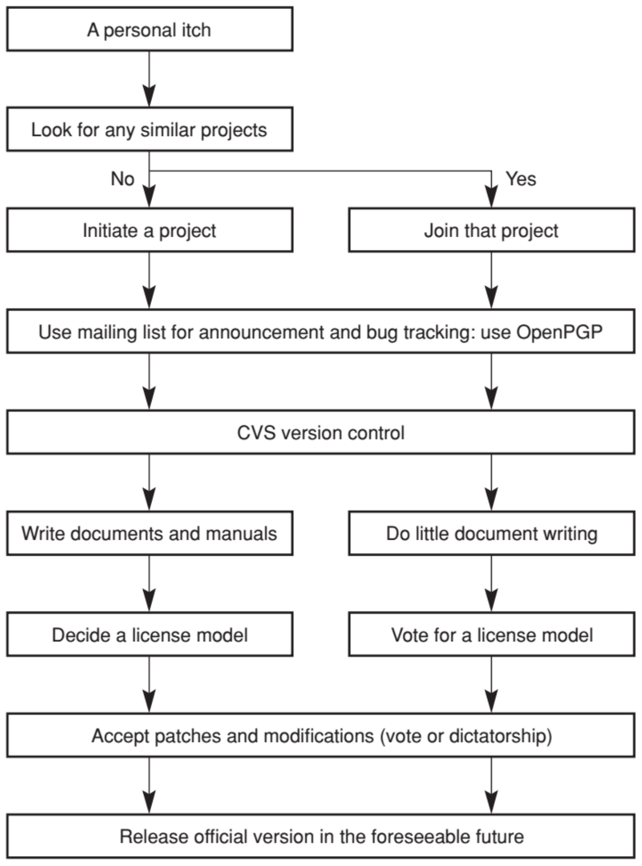
Incremental development processes



The spiral model of the software life-cycle.

Reactive development processes

- **The third form of development cycle is the reactive**, it can be seen as a variation upon the incremental approach. The **development of open source software is probably the best-known example** of this approach.


A personal itch
Look for any similar projects
No
Yes
Initiate a project
Join that project
Use mailing list for announcement and bug tracking: use OpenPGP
CVS version control
Write documents and manuals
Do little document writing
Decide a license model
Vote for a license model
Accept patches and modifications (vote or dictatorship)
Release official version in the foreseeable future
The general open source system development cycle.

Reactive development processes

- Instead of the ideas from a small number of decision-makers (designers) directing the efforts of many developers, **the open source model involves a small number of coordinators drawing together the ideas of many participants.**

- **Many projects have begun with a single developer (as did Linux), and it is the extension and evolution that is then influenced by the many**

Reactive development processes

- **The second example of reactive development is that of the 'website'.**
- **Responsibility for the development and maintenance of the elements of these is often widely decentralized within an organization**, with the **'site manager'** being much more concerned with co-ordination of style than with management of content.
- In recognition of this, a discipline of Web Engineering is emerged in order to address the challenges presented by both the rapid rate of technological change and also, the devolved nature of development.

Design Qualities

Quality Concept

- **Quality** is a concept that can rarely be related to any absolutes, and even for manufactured items ideas about quality usually **cannot be usefully measured on any absolute scale**.

- **The best that we can usually do is to rank items on an ordinal scale**, as when we say that we believe the construction of this coffee table is better than that of another, in terms of materials used and the workmanship it exhibits.

Quality Concept

- Since software is such an abstract product, it is perhaps less apt to be influenced by fashion (a debatable point, perhaps, especially in relation to user interface design), but it is hardly surprising that the **concept of quality is equally difficult to define for software.**

- **Software has both static and dynamic attributes**, and so any set of measures that we can develop **should seek to assess both of these, and should recognize the potential for conflict between them.**

Quality Concept

- For any system, **the ultimate measure of quality should normally be that of fitness for purpose**, since the **end product must be capable of performing the task assigned to it**.
- So, **for software systems**, we can reasonably expect to find that the **final implemented system works, and works correctly in all the required situations. That is, it should perform the required tasks in the specified manner, and within the constraints of the specified resources**.
- This concept is related to the ideas of **Verification and Validation**

Assessing design quality

A framework for assessment

1. **Quality concepts** are the abstract ideas that we have about what constitutes **'good' and 'bad' properties of a system**, and **which will need to be assessed by the designer when making decisions about design choices**

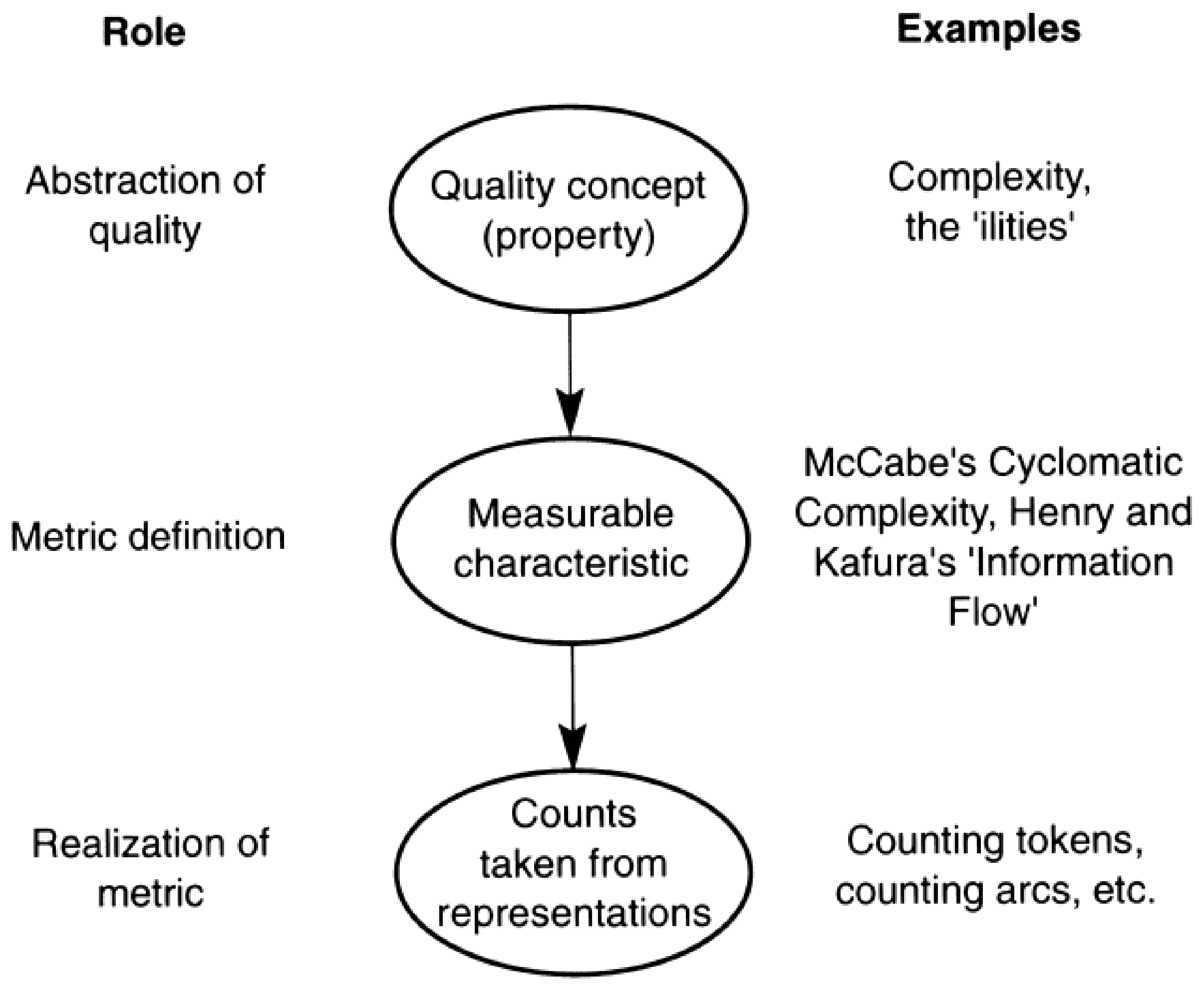
2. **Design attributes** provide a set of measurable characteristics of the design entities and so provide mappings between the abstract idea of a property and the countable features of an actual design (and therefore effectively **correspond to the general concept of a metric**).

Assessing design quality

A framework for assessment

3. **Counts** are concerned with realizing the design attributes, and so involve identifying the lexical features of a representation that will need to be counted in order to obtain some form of values for the metrics

Assessing design quality


Role
Examples
Abstraction of quality
Quality concept (property)
Complexity, the 'ilities'
Metric definition
Measurable characteristic
McCabe's Cyclomatic Complexity, Henry and Kafura's 'Information Flow'
Realization of metric
Counts taken from representations
Counting tokens, counting arcs, etc.

Mapping quality concepts to measurements.

Assessing design quality

A framework for assessment

- As an example of a widely used metric that fits into this framework, **the quality of program code is often considered to be related to its structural complexity.**

- One measurable characteristic that can be used to assess this **complexity is the number of possible control paths** that exist within a program unit, as measured by **McCabe's Cyclomatic Complexity measure**

Assessing design quality

A framework for assessment

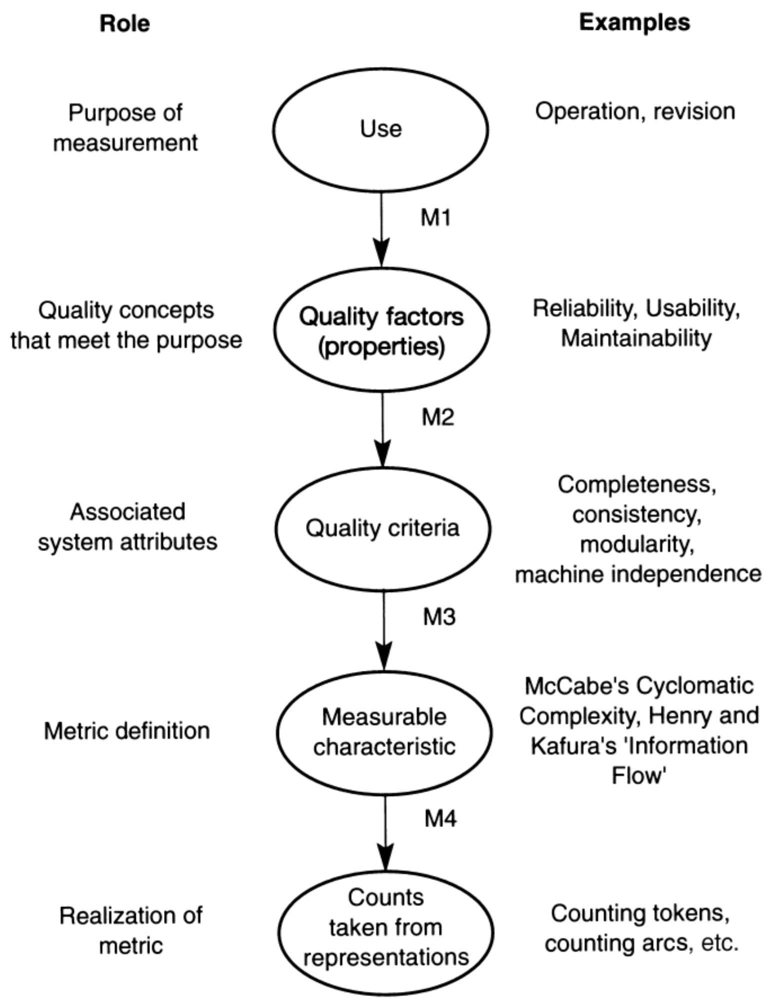
- The **mapping from a measurable characteristic to a set of good and unambiguous counting rules** to be used with a particular implementation language is essentially one of transformation.

- However, the mapping from a quality concept to a set of measurable characteristics is much more a case of making an interpretation

- **To help with this interpretation, we need a fuller model to help identify the mappings required. An expanded form of the framework provided in Figure,**

Assessing design quality

Fuller Mapping


Role
Examples
Purpose of measurement
Use
Operation, revision
M1
Quality concepts that meet the purpose
Quality factors (properties)
Reliability, Usability, Maintainability
M2
Associated system attributes
Quality criteria
Completeness, consistency, modularity, machine independence
M3
Metric definition
Measurable characteristic
McCabe's Cyclomatic Complexity, Henry and Kafura's 'Information Flow'
M4
Realization of metric
Counts taken from representations
Counting tokens, counting arcs, etc.

A fuller mapping from concepts of quality to countable attributes of a design/ implementation.

Assessing design quality

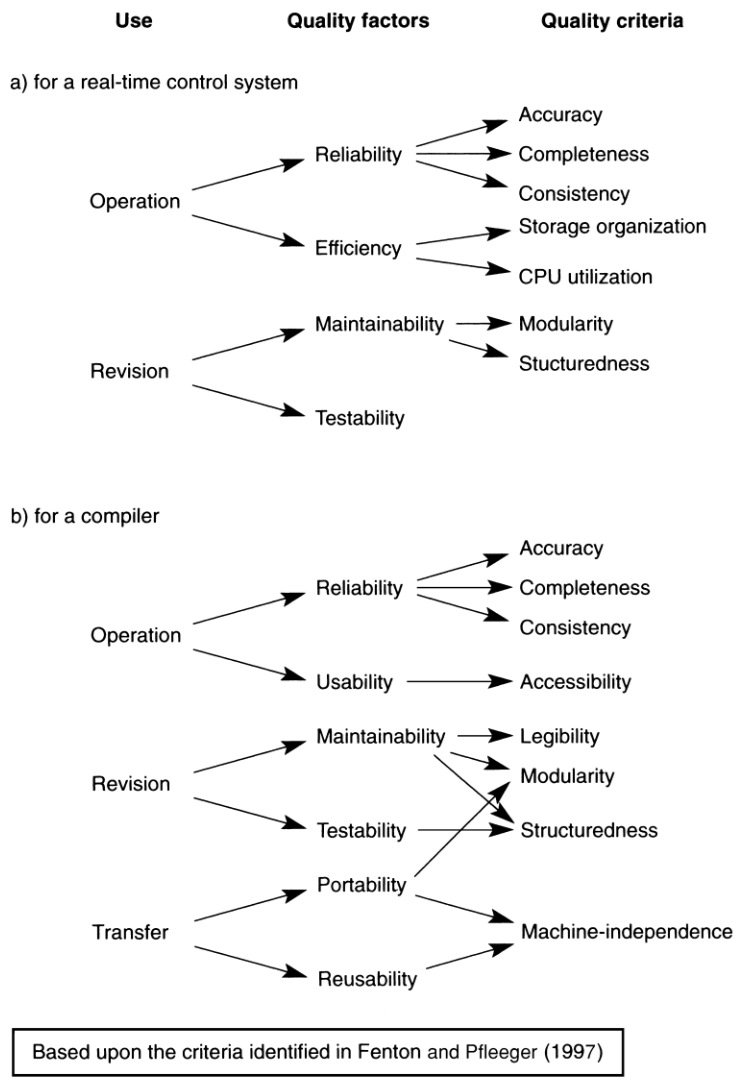
A framework for assessment

- **use** identifies the purpose of making measurements (and hence **strongly differentiates between measuring static properties of a design and measuring dynamic behavioural qualities**);

- **quality factors** determine the **quality concepts** that are associated with the purpose (these items are often referred to as the 'ilities');

- **quality criteria** relate the requirements-oriented properties of the intended system (the 'ilities') to the solution-oriented properties of the design itself, and these are then mapped onto a chosen set of metrics.

Assessing design quality



Mapping quality factors to quality criteria.

Assessing design quality

The ‘ilities’

- The ‘ilities’ form a group of quality factors that need to be considered when making any attempt to assess design quality;

1. **Reliability**
2. **Efficiency**
3. **Maintainability**
4. **Usability**

The other ilities, such as **testability, portability, reusability** and so on are rather more specialized in purpose

Assessing design quality

The ‘ilities’

1. **Reliability**

This factor is essentially concerned with the dynamic characteristics of the eventual system, and so involves the designer in making predictions about behavioural issues. In particular, for the purpose of design we are concerned with ways of determining whether the eventual system will be:

a) **complete**, in the sense of being able to handle all combinations of events and system states;

b) **consistent**, in that its behaviour will be as expected, and will be repeatable, regardless of the overall system loading at any time;

c) **robust** when faced with component failure or some similar conflict

Assessing design quality

The ‘ilities’

2. **Efficiency**

- The efficiency of a system **can be measured through its use of resources such as processor time, memory, network access, system facilities, disk space and so on**

- **As a design factor, efficiency is a difficult property to handle**, since it involves making projections from the design in order to estimate the effects of choices upon eventual resource needs.

Assessing design quality

The 'ilities'

3. **Maintainability**

- As systems get larger and more costly, the need to offset this by ensuring a long life time in service increases in parallel. To help to achieve this**, designs must allow for future modification.**

- Many of the factors that affect maintainability are related to implementation factors or, at least, to very detailed design issues**. Examples of these are the choice of identifiers, comment structuring practices, and documentation standards**

- Designers can help the future maintainers to gain a clear understanding of their original 'mental models'

Assessing design quality

The 'ilities'

4. **Usability**

- There are many issues that can affect usability, but for many systems the design of the user interface (usually termed the Human-Computer Interaction, or HCI) will form an important component, and will influence other design decisions about such features as module boundaries and data structures.

Assessing design quality

Cognitive Dimensions

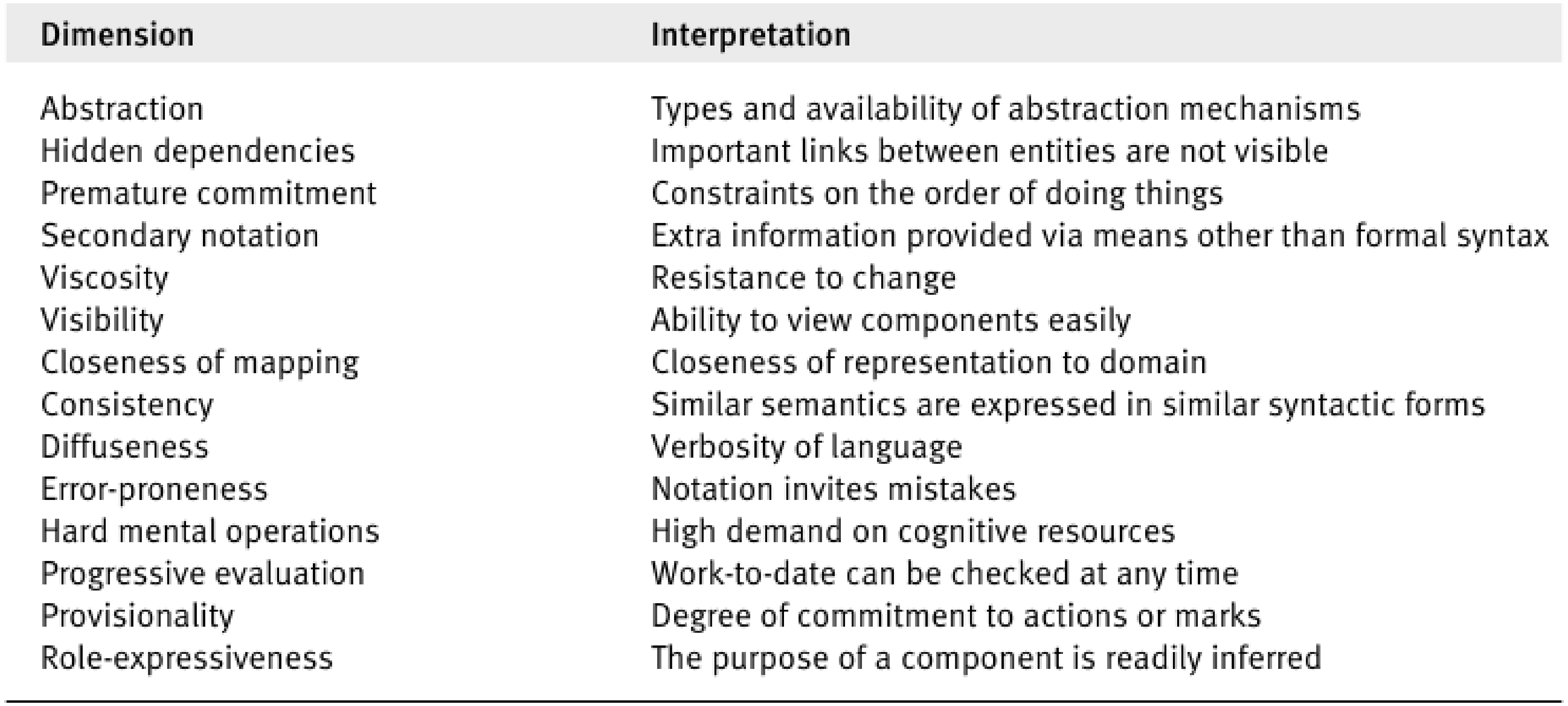
- The 'ilities' are by no means the only framework that we can employ for thinking about system properties within a quality context. **A useful set of concepts that originated in the field of HCI (Human-Computer Interaction) is provided by Green's Cognitive Dimensions framework.**

- While the ideas concerned have been embedded largely in the terminology of HCI and of the more visual aspects of systems, they have nevertheless something useful to offer to the more general process of design

Assessing design quality

Cognitive Dimensions

- Following Table summarizes the cognitive dimensions and their meanings.



Dimension	Interpretation
Abstraction	Types and availability of abstraction mechanisms
Hidden dependencies	Important links between entities are not visible
Premature commitment	Constraints on the order of doing things
Secondary notation	Extra information provided via means other than formal syntax
Viscosity	Resistance to change
Visibility	Ability to view components easily
Closeness of mapping	Closeness of representation to domain
Consistency	Similar semantics are expressed in similar syntactic forms
Diffuseness	Verbosity of language
Error-proneness	Notation invites mistakes
Hard mental operations	High demand on cognitive resources
Progressive evaluation	Work-to-date can be checked at any time
Provisionality	Degree of commitment to actions or marks
Role-expressiveness	The purpose of a component is readily inferred

Assessing design quality

Cognitive Dimensions

1. Premature commitment (and enforced lookahead)

- This arises when an early design decision which was made before proper information was available has the effect of constraining later ones.

- It may be unavoidable (information may not be available within a practical timescale), but it may also arise through the use of procedural design practices such as design methods that encourage a particular ordering in the decision-making process.

- A life-cycle which allows design decisions to be revisited may help, as will a design process which recognizes the need to make several iterations through the decision-making steps.

Assessing design quality

Cognitive Dimensions

2. **Hidden dependencies**

- This concept describes a relationship between two components, such that while one is dependent upon the other the relationship is not fully visible

- Notations often capture one type of dependency (for example, that component A invokes component B) but say nothing about other dependencies that exist between them (A and B share knowledge about certain data types and structures).

- Indeed, the problem becomes potentially much more significant when a design is being modified during maintenance: changing a design is likely to be much more difficult if the designer has only a part of the necessary knowledge about dependencies directly available to them

Assessing design quality

Cognitive Dimensions

3. **Secondary notation**

- This dimension **describes additional information that is conveyed by means other than the 'official syntax'.**

- **Example: Designers make use of local layout conventions when using particular design notations.** While these may be familiar to the originators of the design, **they may not be so evident to others who join the team, or to those who later need to maintain the resulting system.**

Assessing design quality

Cognitive Dimensions

4. Viscosity

- **This describes the concept of 'resistance to change'.**

- **During design development, this may well arise as a consequence of premature commitment**, and **during maintenance it may be a significant reflection of the resulting design 'quality'.**

- **Both conventional software and websites have many mechanisms that can easily result in high viscosity (hard to change)**, especially at the more detailed levels of implementation

- Viscosity is a readily recognizable property at many levels of abstraction, even if it is not one that is easily quantified in design terms. **For a design it may represent the extent to which modules are interdependent, and the differences that exist between the 'conceptual' form of a solution and the one that actually emerges from the design process.**

Quality attributes of the design product

Two principal problems that they present for measurement are:

- **to identify a set of design attributes** that are related to these properties;
- **to find ways of extracting information about these attributes from the available forms of design document**

Some design attributes are

1. Simplicity
2. Modularity
3. Information-hiding

Quality attributes of the design product

1. Simplicity

- **Characteristic of almost all good designs**, in whatever sphere of activity they are produced, is a basic simplicity.

- **A good design meets its objectives and has no additional add-ons that detract from its main purpose**

- One important aid to achieving simplification is abstraction, but it is necessary to use an abstraction that preserves the relevant attributes if it is to be of any help in solving the problem

- **While simplicity cannot easily be assessed**, one can at least seek measures for its **converse characteristic of complexity**

Quality attributes of the design product

2. **Modularity**

- **The use of an appropriate form of modular structuring makes it possible for a given problem to be considered in terms of a set of smaller components.**

- To make good use of a modular structure, one needs to adopt a **design practice based on a separation of concerns.** (**a designer needs to group functions within modules in such a way that their interdependence is minimized**)

- Such a grouping results in a less complex interface to the module: in the case of hardware, it may simply be that fewer interconnections are required; for soft ware, we need to find other criteria in order to measure this factor.

Quality attributes of the design product

2. **Modularity**

Modularity applied in solving a given problem provides at least the **following benefits**:

- modules are **easy to replace**;
- each module captures one feature of a problem, so aiding comprehension (and hence **maintenance**), as well as **providing a framework for designing as a team**;
- a well-structured module can **easily be reused** for another problem

- Two useful quality measures that have long been used for the purpose of assessing the extent of modular structuring in software are **‘coupling’ and ‘cohesion’**

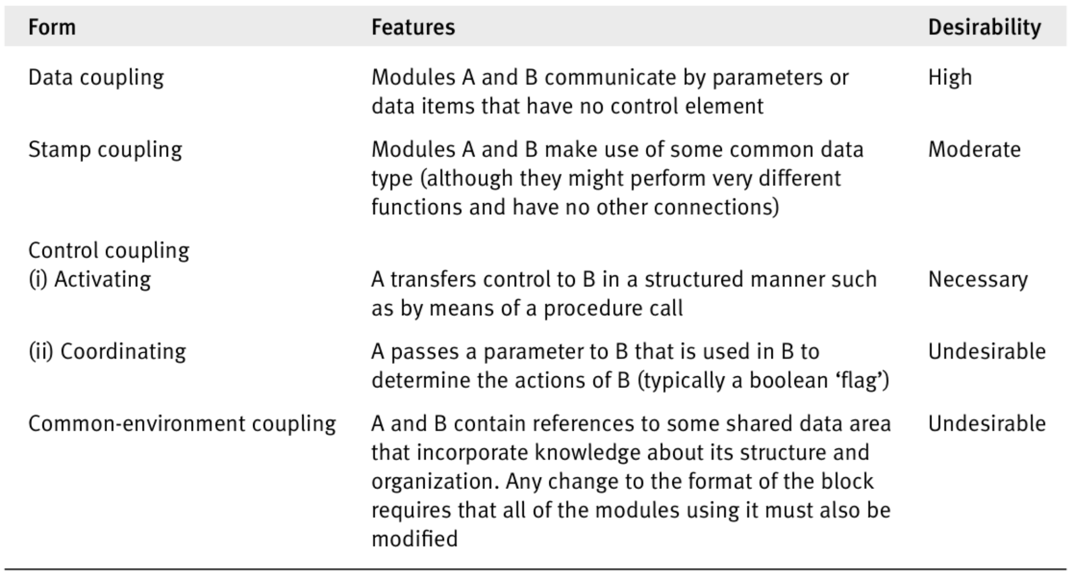
Quality attributes of the design product

2. **Modularity**

'Coupling'

- Coupling **is a measure of intermodule connectivity, and is concerned with identifying the forms of connection that exist between modules and the 'strength' of these connections**.
- Below Table summarizes the principal forms of coupling that are generally regarded as being significant in terms of design features.

Forms of module coupling.



Form	Features	Desirability
Data coupling	Modules A and B communicate by parameters or data items that have no control element	High
Stamp coupling	Modules A and B make use of some common data type (although they might perform very different functions and have no other connections)	Moderate
Control coupling		
(i) Activating	A transfers control to B in a structured manner such as by means of a procedure call	Necessary
(ii) Coordinating	A passes a parameter to B that is used in B to determine the actions of B (typically a boolean 'flag')	Undesirable
Common-environment coupling	A and B contain references to some shared data area that incorporate knowledge about its structure and organization. Any change to the format of the block requires that all of the modules using it must also be modified	Undesirable

Quality attributes of the design product

2. **Modularity**

‘Cohesion’

- Cohesion provides **a measure of the extent to which the components of a module can be considered to be ‘functionally related’**. The ideal module is one in which all the components can be considered as being solely present for one purpose.

- An **example** of functional relatedness might be **a stack class** that contains only a **set of public methods for accessing the stack, together with the data structure for the stack itself**. Such a class can be considered as **exhibiting functional cohesion, since all its components are present solely for the purpose of providing a stack facility.**

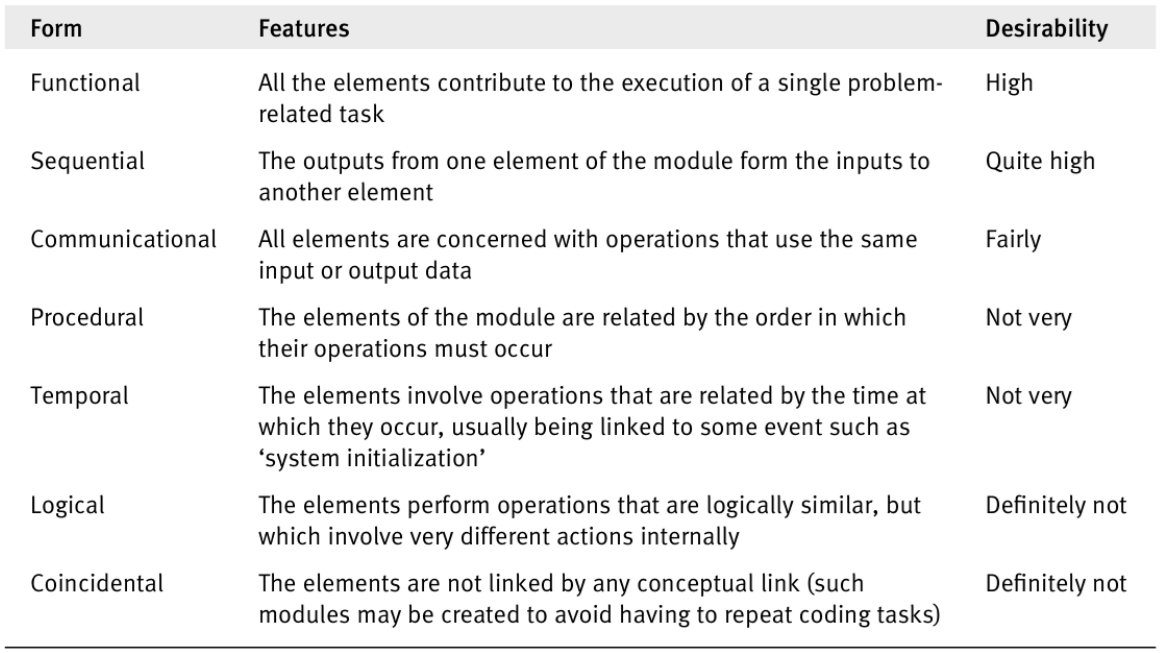
Quality attributes of the design product

2. **Modularity**

‘Cohesion’

- Below Table lists the main forms of cohesion (sometimes termed ‘associations’) that are generally recognized. As can be seen, cohesion is quite easily related to packaging forms of modular structure, as well as to procedural units.

The principal forms of cohesion.



Form	Features	Desirability
Functional	All the elements contribute to the execution of a single problem-related task	High
Sequential	The outputs from one element of the module form the inputs to another element	Quite high
Communicational	All elements are concerned with operations that use the same input or output data	Fairly
Procedural	The elements of the module are related by the order in which their operations must occur	Not very
Temporal	The elements involve operations that are related by the time at which they occur, usually being linked to some event such as ‘system initialization’	Not very
Logical	The elements perform operations that are logically similar, but which involve very different actions internally	Definitely not
Coincidental	The elements are not linked by any conceptual link (such modules may be created to avoid having to repeat coding tasks)	Definitely not

Quality attributes of the design product

2. **Modularity**

‘Cohesion’

- Cohesion is probably even more difficult to quantify than coupling; however, it does provide a measure that can be used to help the designer with assessing the relative merits of possible options.

Quality attributes of the design product

3. **Information Hiding**

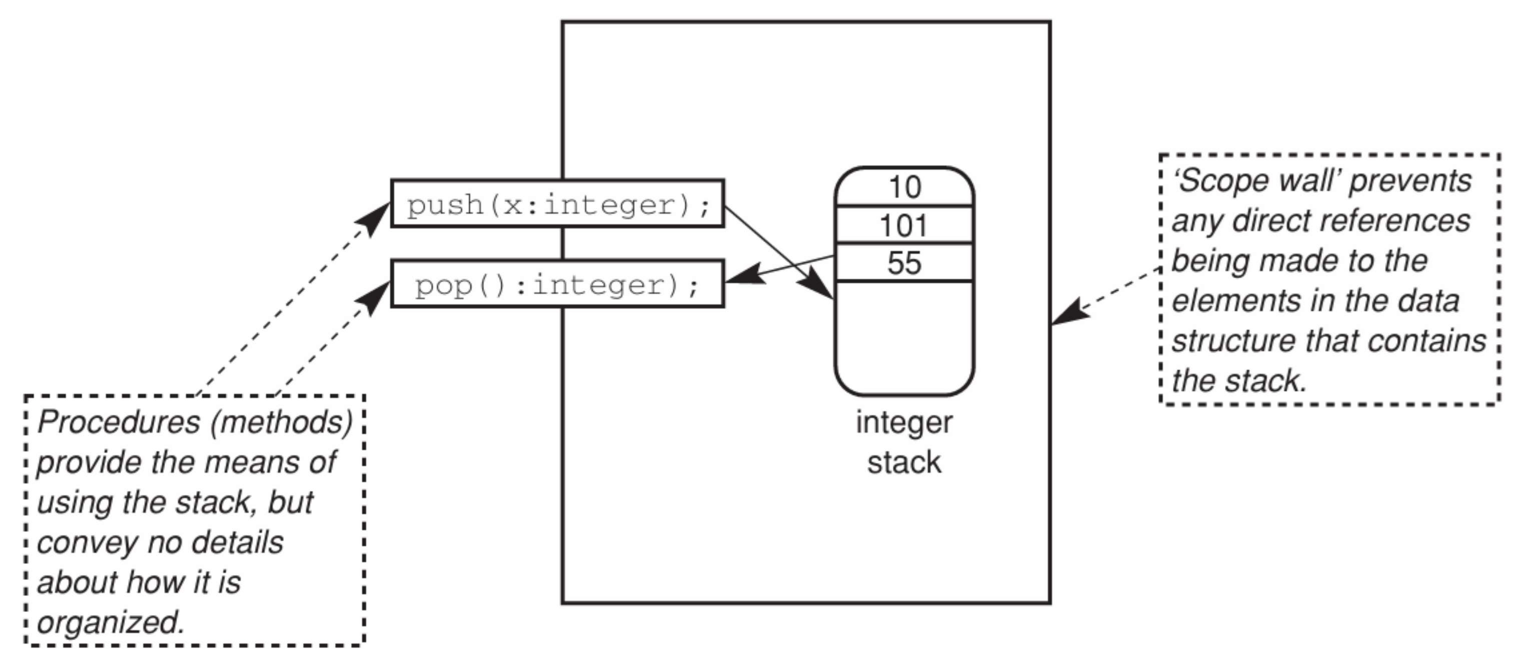
- The basic concept encourages the designer to keep information about the. **detailed forms of such objects as data structures and device interfaces local to a module, or unit, and to ensure that such information should not be made 'visible' outside that unit.**

- Where this principle is applied, **knowledge about the detailed form of any data structures within a module is kept concealed from any other software components that may use the module.**

- As an **example**, this principle might be used in designing a module to provide an integer stack facility to support a set of reverse Polish calculations using integers. **Using the principle of information-hiding, one would seek to conceal the form adopted for implementing the stack, and provide a set of procedures for accessing it**

Quality attributes of the design product

3. **Information Hiding**


push(x:integer);
pop():integer);
10
101
55
integer stack
'Scope wall' prevents any direct references being made to the elements in the data structure that contains the stack.
Procedures (methods) provide the means of using the stack, but convey no details about how it is organized.

Example of information-hiding used with a module that provides an integer stack.

Quality attributes of the design product

Assessing design quality

- A technique that has proved itself useful in assessing design structure and likely behaviour is the **review, or walkthrough** (also sometimes termed an **inspection**)

- **Technical review** – which is concerned with assessing the quality of a design, **Management review** – which is concerned with issues such as project deadlines and schedule.

- Technical reviews can include the use of forms of ‘mental execution’ of the design model, and so can help with assessing dynamic attributes as well as static ones.

Quality attributes of the design product

Assessing design quality

The **eight requirements that they identify for a good design** are that it should be:

1. **well structured**: consistent with chosen properties such as information-hiding;
2. **simple**: to the extent of being ‘as simple as possible, but no simpler’;
3. **efficient**: providing functions that can be computed using the available resources;
4. **adequate**: meeting the stated requirements;
5. **flexible**: able to accommodate likely changes in the requirements, however these might arise;
6. **practical**: module interfaces should provide the required facilities, neither more nor less;
7. **implementable**: using current and available software and hardware technology;
8. **standardized**: using well-defined and familiar notation for any documentation.

Assessing the design process

- The quality of the design process is obviously an important factor that needs to be considered when seeking to understand design quality issues.

- Watts Humphrey (1991) has drawn together a valuable survey of quality issues applied to software development processes. In particular, within his **'Software Process Maturity Model', he identifies five levels of maturity** that occur in the processes that an organization uses for producing software systems.

1. **Initial.** The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success in a project depends upon individual effort.
2. **Repeatable.** Basic project management processes are established and used to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3. **Defined.** The software process for both management and engineering activities is documented, standardized and integrated into an organization-wide software process. All projects use a documented and approved version of the organization's process for developing and maintaining software.
4. **Managed.** Detailed measures of the software process and product quality are collected. Both the software process and the products are quantitatively understood and controlled using detailed measures.
5. **Optimizing.** Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies.

Assessing the design process

- To improve the quality of the design process it is particularly necessary to find ways of including input to design activities that can provide:

1. **Domain knowledge** about the type of problem involved, and about important aspects of any implementation features;

2. **Method knowledge** that helps with understanding any design techniques being used; Assessing the design process

3. **Experience** from similar projects, wherever available.

Assessing the design process

- There are three widely adopted ways of providing these inputs to the design process. :

1. **Technical reviews** is largely aimed at improving the design product, it can also have a significant impact upon the design process by identifying useful techniques or notations that might be applicable to a particular problem;

2. **Management Reviews** are primarily concerned with developing and refining estimates of effort and deadlines for the project as a whole, and with gathering any data that might be needed for such estimates.

3. **Prototyping** By constructing suitable prototypes, it may therefore be possible to supplement the knowledge and experience available to both management and the design team.

Assessing the design process

- The use of **reporting and tracking mechanisms** will play an important role in monitoring quality issues

- **Quality of design documentation** plays an important role in establishing a high-quality process by reusing experience.

THANK YOU