SWE2007
Software Construction and Maintenance Digital Assignment - 2

Name: VEENASREE P
Reg no: 20MIS1139

1. Building your own assertion routine is a good example of programming "into" a language rather than just programming "in" a language.

a. Analyse the above statement with illustrative examples.

b. Explain how assertion can be implemented in different programming languages

a) When using the imperative programming paradigm, an assertion is a predicate (a Boolean-valued function over the state space, usually expressed as a logical proposition using the variables of a program) connected to a point in the program, that always should evaluate to true at that point in code execution. Assertions can help a programmer read the code, help a compiler compile it, or help the program detect its own defects.

For the latter, some programs check assertions by actually evaluating the predicate as they run. Then, if it is not in fact true – an assertion failure – the program considers itself to be broken and typically deliberately crashes or throws an assertion failure exception.

The following code contains two assertions, x > 0 and x > 1, and they are indeed true at the indicated points during execution:

```
x = 1;
assert x > 0;
x++;
assert x > 1;
```

Programmers can use assertions to help specify programs and to reason about program correctness. For example, a precondition—an assertion placed at the beginning of a section of code—determines the set of states under which the

programmer expects the code to execute. A postcondition—placed at the end—describes the expected state at the end of execution. For example: x > 0 { x++ } x > 1.

Assertions in design by contract:

Assertions can function as a form of documentation: they can describe the state the code expects to find before it runs (its preconditions), and the state the code expects to result in when it is finished running (postconditions); they can also specify invariants of a class. Eiffel integrates such assertions into the language and automatically extracts them to document the class. This forms an important part of the method of design by contract.

This approach is also useful in languages that do not explicitly support it: the advantage of using assertion statements rather than assertions in comments is that the program can check the assertions every time it runs; if the assertion no longer holds, an error can be reported. This prevents the code from getting out of sync with the assertions.

Assertions for run-time checking:

An assertion may be used to verify that an assumption made by the programmer during the implementation of the program remains valid when the program is executed.

For example, consider the following Java code:

```
int total = countNumberOfUsers();
if (total % 2 == 0) {
// total is even
}
 else {
// total is odd and non-negative assert total
% 2 == 1;
}
```

b) In Java, % is the remainder operator (modulo), and in Java, if its first operand is negative, the result can also be negative (unlike the modulo used in mathematics). Here, the programmer has assumed that total is non-negative, so that the remainder of a division with 2 will always be 0 or 1. The assertion makes this assumption explicit: if countNumberOfUsers does return a negative value, the program may have a bug.

A major advantage of this technique is that when an error does occur it is detected immediately and directly, rather than later through often obscure effects. Since an assertion failure usually reports the code location, one can often pin-point the error without further debugging.

Assertions are also sometimes placed at points the execution is not supposed to reach. For example, assertions could be placed at the default clause of the switch statement in languages such as C, C++, and Java. Any case which the programmer does not handle intentionally will raise an error and the program will abort rather than silently continuing in an erroneous state. In D such an assertion is added automatically when a switch statement doesn't contain a default clause.

In Java, assertions have been a part of the language since version 1.4. Assertion failures result in raising an AssertionError when the program is run with the appropriate flags, without which the assert statements are ignored. In C, they are added on by the standard header assert.h defining assert (assertion) as a macro that signals an error in the case of failure, usually terminating the program. In C++, both assert.h and cassert headers provide the assert macro.

The danger of assertions is that they may cause side effects either by changing memory data or by changing thread timing. Assertions should be implemented carefully so they cause no side effects on program code.

Assertion constructs in a language allow for easy test-driven development (TDD) without the use of a third-party library.

2. Detail with Illustrative examples about the implementation of defensive programming and Exception and Error handling in two different programming languages.

Defensive Programming:

Defensive programming is a form of defensive design intended to develop programs that are capable of detecting potential security abnormalities and make predetermined responses. It ensures the continuing function of a piece of software under unforeseen circumstances. Defensive programming practices are often used where high availability, safety, or security is needed.

Defensive programming is an approach to improve software and source code, in terms of:

General quality – reducing the number of software bugs and problems.

Making the source code comprehensible – the source code should be readable and understandable so it is approved in a code audit.

Making the software behave in a predictable manner despite unexpected inputs or user actions.

Overly defensive programming, however, may safeguard against errors that will never be encountered, thus incurring run-time and maintenance costs. There is also a risk that code traps prevent too many exceptions, potentially resulting in unnoticed, incorrect results.

Secure programming is the subset of defensive programming concerned with computer security. Security is the concern, not necessarily safety or availability (the software may be allowed to fail in certain ways). As with all kinds of defensive programming, avoiding bugs is a primary objective; however, the motivation is not as much to reduce the likelihood of failure in normal operation (as if safety were the concern), but to reduce the attack surface – the programmer must assume that the software might be misused actively to reveal bugs, and that bugs could be exploited maliciously.

```
int risky_programming(char *input) {
char str[1000];
 // ...
strcpy(str, input); // Copy input.
//…
}
```

Offensive programming is a category of defensive programming, with the added emphasis that certain errors should not be handled defensively. In this practice, only errors from outside the program's control are to be handled (such as user input); the software itself, as well as data from within the program's line of defense, are to be trusted in this methodology.

Trusting internal data validity:

Overly defensive programming

```
const char* trafficlight_colorname(enum traffic_light_color c) {
switch (c) {
case TRAFFICLIGHT_RED: return "red";
case TRAFFICLIGHT_YELLOW: return "yellow";
case TRAFFICLIGHT_GREEN: return "green";
}
return "black";
// To be handled as a dead traffic light.
// Warning: This last 'return' statement will be dropped by an optimizing
// compiler if all possible values of 'traffic_light_color' are listed in
// the previous 'switch' statement...
}
```

Trusting software components:

Overly defensive programming

```
if (is_legacy_compatible(user_config)) {
// Strategy: Don't trust that the new code behaves the same
old_code(user_config);
}
else {
```

```
// Fallback: Don't trust that the new code handles the same cases
if (new_code(user_config) != OK) {
old_code(user_config);
}
}
```

Techniques

Here are some defensive programming techniques:

Intelligent source code reuse

If existing code is tested and known to work, reusing it may reduce the chance of bugs being introduced.

However, reusing code is not always good practice. Reuse of existing code, especially when widely distributed, can allow for exploits to be created that target a wider audience than would otherwise be possible and brings with it all the security and vulnerabilities of the reused code.

When considering using existing source code, a quick review of the modules(sub-sections such as classes or functions) will help eliminate or make the developer aware of any potential vulnerabilities and ensure it is suitable to use in the project.

Exception and Error Handling:

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

Exceptions occur for numerous reasons, including invalid user input, code errors, device failure, the loss of a network connection, insufficient memory to run an application, a memory conflict with another program, a program attempting to divide by zero or a user attempting to open files that are unavailable.

When an exception occurs, specialized programming language constructs, interrupt hardware mechanisms or operating system interprocess communication facilities handle the exception.

Exception handling differs from error handling in that the former involves conditions an application might catch versus serious problems an application might want to avoid. In contrast, error handling helps maintain the normal flow of software program execution.

How is exception handling used?

If a program has a lot of statements and an exception happens halfway through its execution, the statements after the exception do not execute, and the program crashes. Exception handling helps ensure this does not happen when an exception occurs.

Exception handling can catch and throw exceptions. If a detecting function in a block of code cannot deal with an anomaly, the exception is thrown to a function that can handle the exception. A catch statement is a group of statements that handle the specific thrown exception. Catch parameters determine the specific type of exception that is thrown.

Exception handling is useful for dealing with exceptions that cannot be handled locally. Instead of showing an error status in the program, the exception handler transfers control to where the error can be handled. A function can throw exceptions or can choose to handle exceptions.

Error handling code can also be separated from normal code with the use of try blocks, which is code that is enclosed in curly braces or brackets that could cause an exception. Try blocks can help programmers to categorize exception objects.

Types of Exception classes:

1. Checked exceptions
2. Unchecked exceptions

Error handling code can also be separated from normal code with the use of try blocks, which is code that is enclosed in curly braces or brackets that could cause an exception. Try blocks can help programmers to categorize exception objects.

Exception handling in Java

The try bracket contains the code that encounters the exception and prevents the application from crashing.

Exception handling in Java vs. exception handling in C++:

Although the try, throw and catch blocks are all the same in the Java and C++ programming languages, there are some basic differences in each language.

For example, C++ exception handling has a catch all block, which can catch different types of exceptions, but Java does not. Likewise, C++ is able to throw primitives and pointers as exceptions, but Java can only throw objects as exceptions.
Unlike C++, Java has both checked and unchecked exceptions. Java also has a finally clause, which executes after the try-catch block for cleanup. C++ does not have a finally block. However, the finalize method will be removed in future versions of Java, which means users will have to find different methods to handle Java errors and cleanup.

Examples of exception handling

The following are examples of exceptions:

SQLException is a checked exception that occurs while executing queries on a database for Structured Query Language syntax.

ClassNotFoundException is a checked exception that occurs when the required class is not found -- either due to a command-line error, a missing CLASS file or an issue with the classpath.

IllegalStateException is an unchecked exception that occurs when an environment's state does not match the operation being executed.

IllegalArgumentException is an unchecked exception that occurs when an incorrect argument is passed to a method.

NullPointerException is an unchecked exception that occurs when a user tries to access an object using a reference variable that is null or empty.

In this example, a variable is left undefined, so console.log generates an exception. The try bracket is used to contain the code that encounters the exception, so the application does not crash. The catch block is skipped if the code works. But, if an exception occurs, then the error is caught, and the catch block is executed.

3. With Illustrative examples discuss the various (Minimum FIVE) design practices and their associated design challenges.

1) Design and Construction as an Integrated System:

In the planning of facilities, it is important to recognize the close relationship between design and construction. These processes can best be viewed as an integrated system. Broadly speaking, design is a process of creating the description of a new facility, usually represented by detailed plans and specifications; construction planning is a process of identifying activities and resources required to make the design a physical reality. Hence, construction is the implementation of a design envisioned by architects and engineers. In both design and construction, numerous operational tasks must be performed with a variety of precedence and other relationships among the different tasks.

Several characteristics are unique to the planning of constructed facilities and should be kept in mind even at the very early stage of the project life cycle. These include the following:

- Nearly every facility is custom designed and constructed, and often requires a long time to complete.
- Both the design and construction of a facility must satisfy the conditions peculiar to a specific site.
- Because each project is site specific, its execution is influenced by natural, social and other locational conditions such as weather, labor supply, local building codes, etc.
- Since the service life of a facility is long, the anticipation of future requirements is inherently difficult.
- Because of technological complexity and market demands, changes of design plans during construction are not uncommon.

In an integrated system, the planning for both design and construction can proceed almost simultaneously, examining various alternatives which are desirable from both viewpoints and thus eliminating the necessity of extensive revisions under the guise of value engineering. Furthermore, the review of designs with regard to their constructibility can be carried out as the project progresses from planning to design.

For example, if the sequence of assembly of a structure and the critical loadings on the partially assembled structure during construction are carefully considered as a part of the overall structural design, the impacts of the design on construction falsework and on assembly details can be anticipated. However, if the design professionals are expected to assume such responsibilities, they must be rewarded for sharing the risks as well as for undertaking these additional tasks.

Similarly, when construction contractors are expected to take over the responsibilities of engineers, such as devising a very elaborate scheme to erect an unconventional structure, they too must be rewarded accordingly. As long as the owner does not assume the responsibility for resolving this risk-reward dilemma, the concept of a truly integrated system for design and construction cannot be realized. It is interesting to note that European owners are generally more open to new technologies and to share risks with designers and contractors.

In particular, they are more willing to accept responsibilities for the unforeseen subsurface conditions in geotechnical engineering. Consequently, the designers and contractors are also more willing to introduce new techniques in order to reduce the time and cost of construction. In European practice, owners typically present contractors with a conceptual design, and contractors prepare detailed designs, which are checked by the owner's engineers. Those detailed designs may be alternate designs, and specialty contractors may also prepare detailed alternate designs.

2) Innovation and Technological Feasibility:

The planning for a construction project begins with the generation of concepts for a facility which will meet market demands and owner needs. Innovative concepts in design are highly valued not for their own sake but for their contributions to reducing costs and to the improvement of aesthetics, comfort or convenience as embodied in a well-designed facility. However, the constructor as well as the design professionals must have an appreciation and full understanding of the technological complexities often associated with innovative designs in order to provide a safe and sound facility. Since these concepts are often preliminary or tentative, screening studies are carried out to determine the overall technological viability and economic attractiveness without pursuing these concepts in great detail. Because of the ambiguity of the objectives and the uncertainty of external events, screening studies call for

uninhibited innovation in creating new concepts and judicious judgment in selecting the appropriate ones for further consideration.

Innovative design concepts must be tested for technological feasibility. Three levels of technology are of special concern: technological requirements for operation or production, design resources and construction technology. The first refers to the new technologies that may be introduced in a facility which is used for a certain type of production such as chemical processing or nuclear power generation. The second refers to the design capabilities that are available to the designers, such as new computational methods or new materials. The third refers to new technologies which can be adopted to construct the facility, such as new equipment or new construction methods.

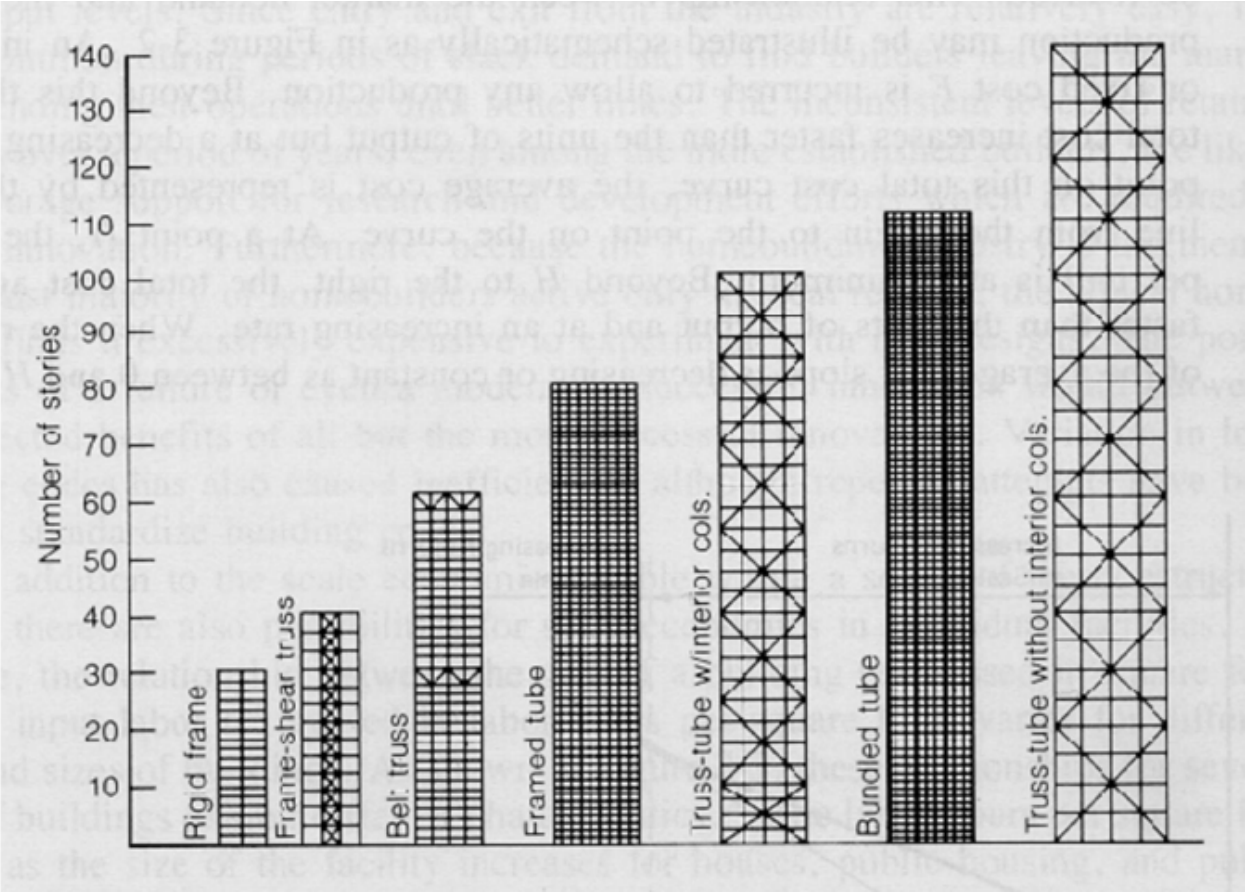
A new facility may involve complex new technology for operation in hostile environments such as severe climate or restricted accessibility. Large projects with unprecedented demands for resources such as labour supply, material and infrastructure may also call for careful technological feasibility studies. Major elements in a feasibility study on production technology should include, but are not limited to, the following:

- Project type as characterized by the technology required, such as synthetic fuels, petrochemicals, nuclear power plants, etc.
- Project size in dollars, design engineer's hours, construction labor hours, etc.
- Design, including sources of any special technology which require licensing agreements.
- Project location which may pose problems in environmental protection, labour productivity and special risks.

An example of innovative design for operation and production is the use of entropy concepts for the design of integrated chemical processes. Simple calculations can be used to indicate the minimum energy requirements and the least number of heat exchange units to achieve desired objectives. The result is a new incentive and criterion for designers to achieve more effective designs. Numerous applications of the new methodology has shown its efficacy in reducing both energy costs and construction expenditures. This is a case in which innovative design is not a matter of trading-off operating and capital costs, but better designs can simultaneously achieve improvements in both objectives.

The structural design of skyscrapers offers an example of innovation in overcoming the barrier of high costs for tall buildings by making use of new design capabilities. A revolutionary concept in skyscraper design was introduced in the 1960's by Fazlur Khan who argued that, for a building of a given height, there is an appropriate structural system which would produce the most efficient use of the material.

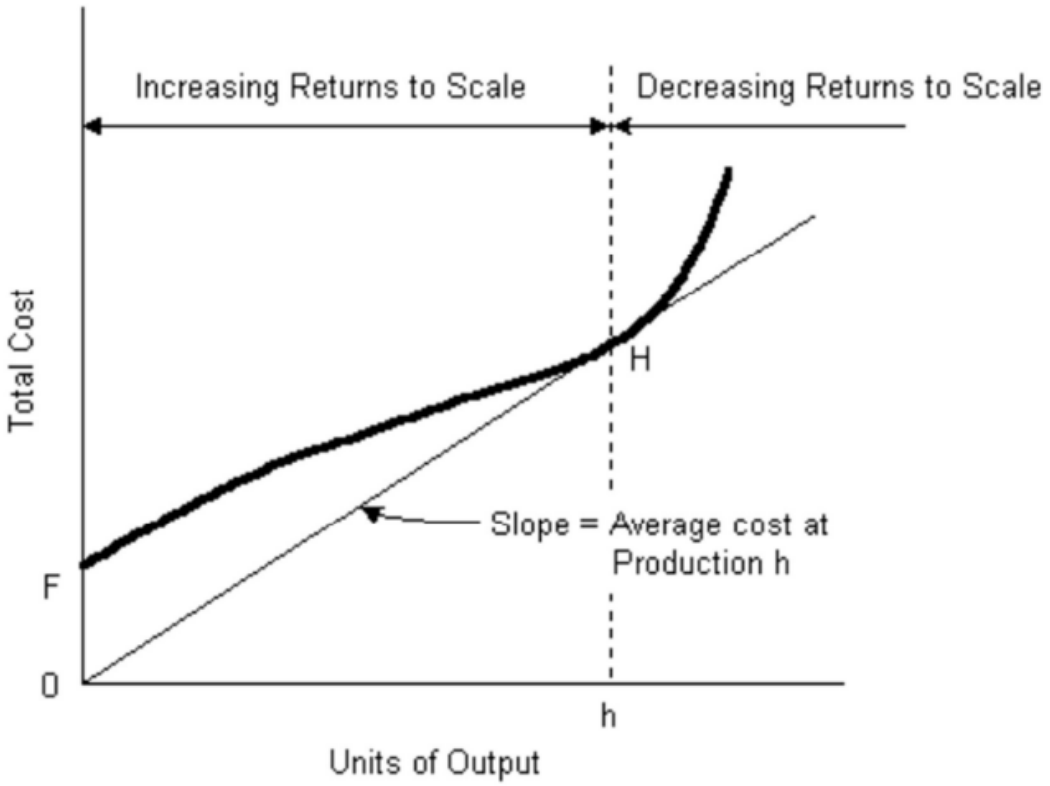
Before 1965, most skyscrapers were steel rigid frames. However, Fazlur Khan believed that it was uneconomical to construct all office buildings of rigid frames, and proposed an array of appropriate structural systems for steel buildings of specified heights as shown in Figure 3-1. By choosing an appropriate structural system, an engineer can use structural materials more efficiently. For example, the 60-story Chase Manhattan Building in New York used about 60 pounds per square foot of steel in its rigid frame structure, while the 100-story John Hancock Center in Chicago used only 30 pounds per square foot for a trusted tube system. At the time the Chase Manhattan Building was constructed, no bracing was used to stiffen the core of a rigid frame building because design engineers did not have the computing tools to do the complex mathematical analysis associated with core bracing.



3) Innovation and Economic Feasibility:

Innovation is often regarded as the engine which can introduce construction economies and advance labor productivity. This is obviously true for certain types of innovations in industrial production technologies, design capabilities, and construction equipment and methods. However, there are also limitations due to the economic infeasibility of such innovations, particularly in the segments of construction industry which are more fragmented and permit ease of entry, as in the construction of residential housing.

Market demand and firm size play an important role in this regard. If a builder is to construct a larger number of similar units of buildings, the cost per unit may be reduced. This relationship between the market demand and the total cost of production may be illustrated schematically as in Figure 3-2. An initial threshold or fixed cost F is incurred to allow any production. Beyond this threshold cost, total cost increases faster than the units of output but at a decreasing rate. At each point on this total cost curve, the average cost is represented by the slope of a line from the origin to the point on the curve. At a point H, the average cost per unit is at a minimum. Beyond H to the right, the total cost again increases faster than the units of output and at an increasing rate. When the rate of change of the average cost slope is decreasing or constant as between 0 and H on the curve, the range between 0 and H is said to be increasing return to scale; when the rate of change of the average cost slope is increasing as beyond H to the right, the region is said to be decreasing return to scale. Thus, if fewer than h units are constructed, the unit price will be higher than that of exactly h units. On the other hand, the unit price will increase again if more than h units are constructed.

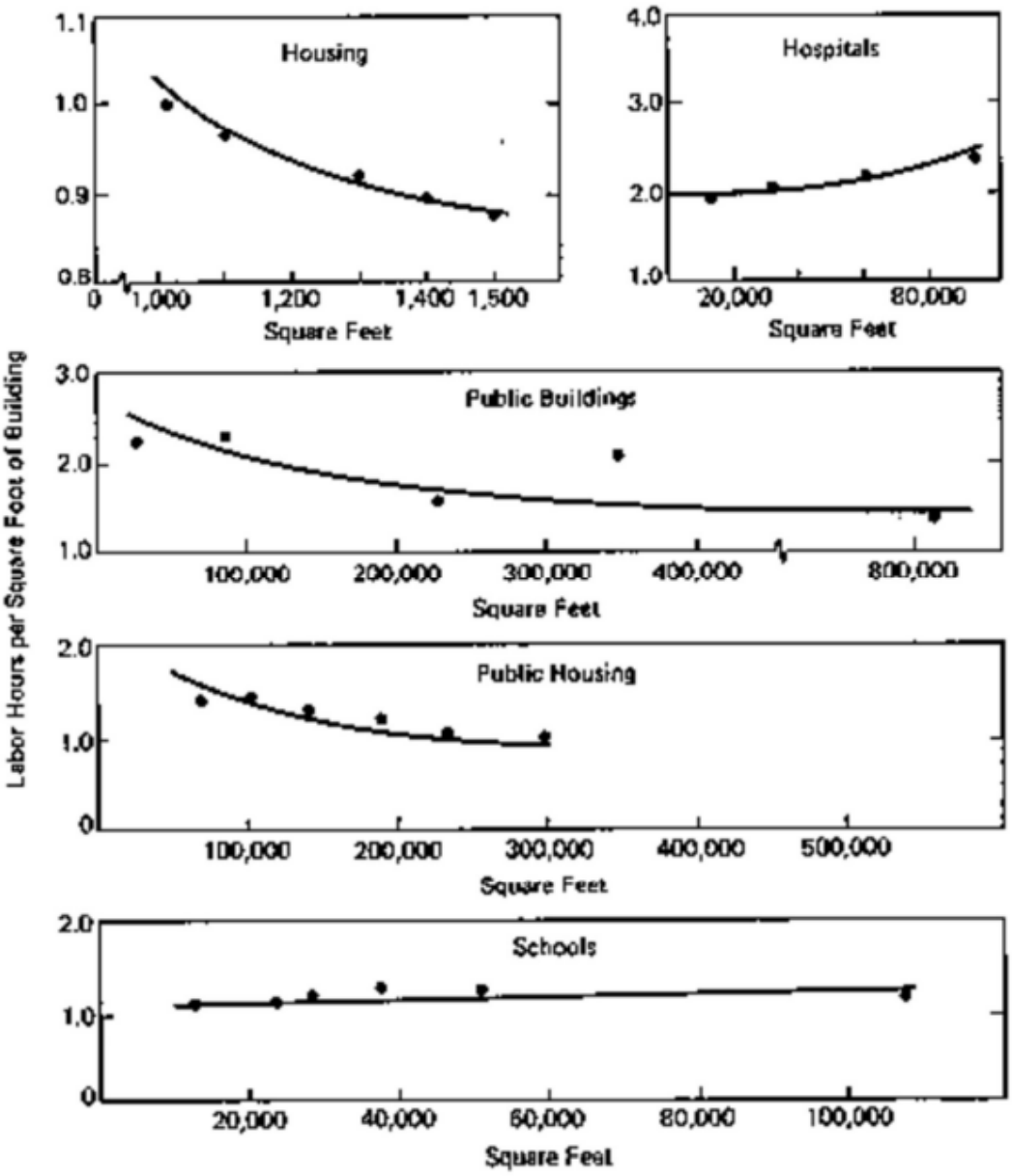


Nowhere is the effect of market demand and total cost more evident than in residential housing. The housing segment in the last few decades accepted many innovative technical improvements in building materials which were promoted by material suppliers. Since material suppliers provide products to a large number of homebuilders and others, they are in a better position to exploit production economies of scale and to support new product development. However, homebuilders themselves have not been as successful in making the most fundamental form of innovation which encompasses changes in the technological process of homebuilding by shifting the mixture of labour and material inputs, such as substituting large scale off-site prefabrication for on-site assembly.

There are several major barriers to innovation in the technological process of homebuilding, including demand instability, industrial fragmentation, and building codes. Since market demand for new homes follows demographic trends and other socio-economic conditions, the variation in home building has been anything but regular. The profitability of the homebuilding industry has closely matched aggregate output levels. Since entry and exist from the industry are

relatively easy, it is not uncommon during periods of slack demand to find builders leaving the market or suspending their operations until better times. The inconsistent levels of retained earnings over a period of years, even among the more established builders, are likely to discourage support for research and development efforts which are required to nurture innovation. Furthermore, because the homebuilding industry is fragmented with a vast majority of homebuilders active only in local regions, the typical homebuilder finds it excessively expensive to experiment with new designs. The potential costs of a failure or even a moderately successful innovation would outweigh the expected benefits of all but the most successful innovations. Variation in local building codes has also caused inefficiencies although repeated attempts have been made to standardize building codes.

In addition to the scale economies visible within a sector of the construction market, there are also possibilities for scale economies in individual facility. For example, the relationship between the size of a building (expressed in square feet) and the input labour (expressed in labour hours per square foot) varies for different types and sizes of buildings. As shown in Figure 3-3, these relationships for several types of buildings exhibit different characteristics. The labour hours per square foot decline as the size of facility increases for houses, public housing and public buildings. However, the labour hours per square foot almost remains constant for all sizes of school buildings and increases as the size of a hospital facility increases.



4) Design Methodology:

While the conceptual design process may be formal or informal, it can be characterized by a series of actions: formulation, analysis, search, decision, specification, and modification. However, at the early stage in the development of a new project, these actions are highly interactive as illustrated in Figure 3-4. Many iterations of redesign are expected to refine the functional requirements, design concepts and financial constraints, even though the analytic tools applied to the solution of the problem at this stage may be very crude.

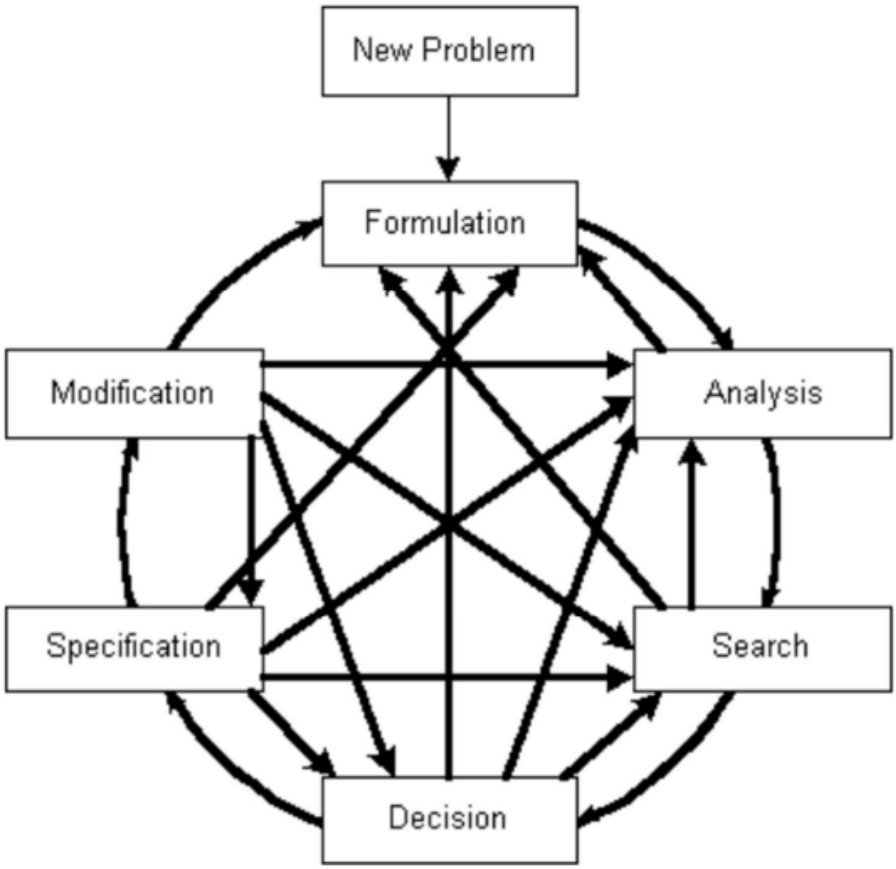


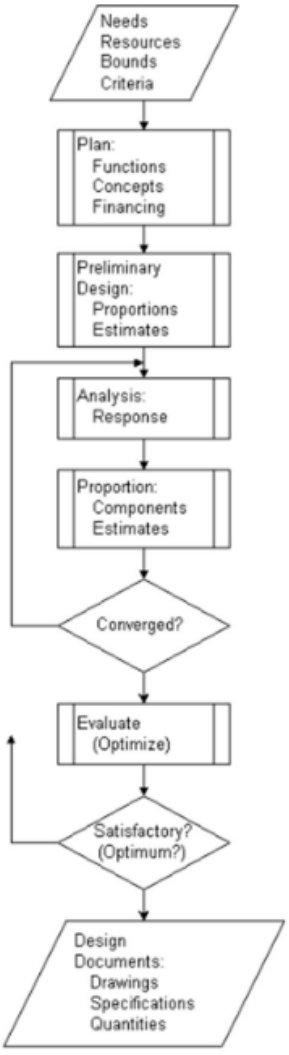
Figure 3-4: Conceptual Design Process

The series of actions taken in the conceptual design process may be described as follows:

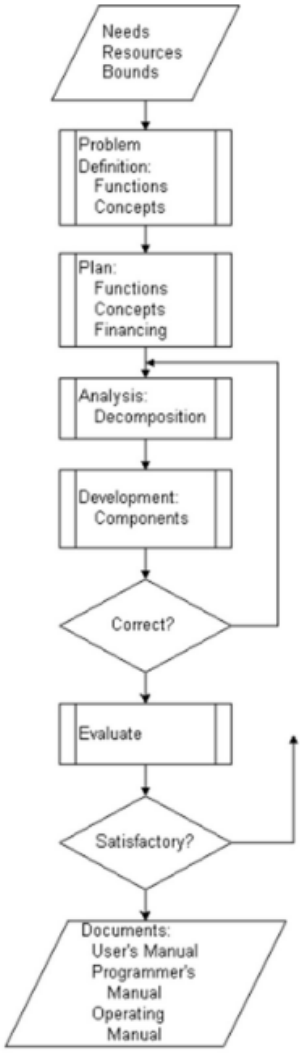
- Formulation refers to the definition or description of a design problem in broad terms through the synthesis of ideas describing alternative facilities.
- Analysis refines the problem definition or description by separating important from peripheral information and by pulling together the essential detail. Interpretation and prediction are usually required as part of the analysis.
- Search involves gathering a set of potential solutions for performing the specified functions and satisfying the user requirements.
- Decision means that each of the potential solutions is evaluated and compared to the alternatives until the best solution is obtained.
- Specification is to describe the chosen solution in a form which contains enough detail for implementation.

- Modification refers to the change in the solution or re-design if the solution is found to be wanting or if new information is discovered in the process of design.

As the project moves from conceptual planning to detailed design, the design process becomes more formal. In general, the actions of formulation, analysis, search, decision, specification and modification still hold, but they represent specific steps with less random interactions in detailed design. The design methodology thus formalized can be applied to a variety of design problems. For example, the analogy of the schematic diagrams of the structural design process and of the computer program development process is shown in Figure



Schematic diamgram of structural design process.



Schematic diagram of computer program development process.

The basic approach to design relies on decomposition and integration. Since design problems are large and complex, they have to be decomposed to yield subproblems that are small enough to solve. There are numerous alternative ways to decompose design problems, such as decomposition by functions of the facility, by spatial locations of its parts, or by links of various functions or parts. Solutions to subproblems must be integrated into an overall solution. The integration often creates conceptual conflicts which must be identified and corrected. A hierarchical structure with an appropriate number of levels may be used for the decomposition of a design problem to subproblems.

For example, in the structural design of a multistorey building, the building may be decomposed into floors, and each floor may in turn be decomposed into separate areas. Thus, a hierarchy representing the levels of building, floor and area is formed.

Different design styles may be used. The adoption of a particular style often depends on factors such as time pressure or available design tools, as well as the nature of the design problem. Examples of different styles are:

- Top-down design: Begin with a behaviour description of the facility and work towards descriptions of its components and their interconnections.
- Bottom-up design: Begin with a set of components, and see if they can be arranged to meet the behaviour description of the facility.

The design of a new facility often begins with the search of the files for a design that comes as close as possible to the one needed. The design process is guided by accumulated experience and intuition in the form of heuristic rules to find acceptable solutions. As more experience is gained for this particular type of facility, it often becomes evident that parts of the design problem are amenable to rigorous definition and algorithmic solution. Even formal optimization methods may be applied to some parts of the problem.

5) Functional Design:

The objective of functional design for a proposed facility is to treat the facility as a complex system of interrelated spaces which are organized systematically according to the functions to be performed in these spaces in order to serve a collection of needs. The arrangement of physical spaces can be viewed as an iterative design process to find a suitable floor plan to facilitate the movement of people and goods associated with the operations intended.

A designer often relies on a heuristic approach, i.e., applying selected rules or strategies serving to stimulate the investigation in search for a solution. The heuristic approach used in arranging spatial layouts for facilities is based generally on the following considerations:

1. Identification of the goals and constraints for specified tasks,
2. Determination of the current state of each task in the iterative design process,
3. Evaluation of the differences between the current state and the goals,
4. Means of directing the efforts of search towards the goals on the basis of past experience.

Hence, the procedure for seeking the goals can be recycled iteratively in order to make tradeoffs and thus improve the solution of spatial layouts.

Consider, for example, an integrated functional design for a proposed hospital. Since the responsibilities for satisfying various needs in a hospital are divided among different groups of personnel within the hospital administrative structure, a hierarchy of functions corresponding to different levels of responsibilities is proposed in the systematic organization of hospital functions. In this model, the functions of a hospital system are decomposed into a hierarchy of several levels:

1. Hospital--conglomerate of all hospital services resulting from top policy decisions,
2. Division--broadly related activities assigned to the same general area by administrative decisions,
3. Department--combination of services delivered by a service or treatment group,
4. Suite--specific style of common services or treatments performed in the same suite of rooms,
5. Room--all activities that can be carried out in the same internal environment surrounded by physical barriers,
6. Zone--several closely related activities that are undertaken by individuals,
7. Object--a single activity associated with an individual.

In the integrated functional design of hospitals, the connection between physical spaces and functions is most easily made at the lowest level of the hierarchy, and then extended upward to the next higher level. For example, a bed is a physical object immediately related to the activity of a patient. A set of furniture consisting of a bed, a night table and an armchair arranged comfortably in a zone indicates the sphere of private activities for a patient in a room with multiple occupancy.

Thus, the spatial representation of a hospital can be organized in stages starting from the lowest level and moving to the top. In each step of the organization process, an element (space or function) under consideration can be related directly to the elements at the levels above it, to those at the levels below it, and to those within the same level.

Since the primary factor relating spaces is the movement of people and supplies, the objective of arranging spaces is the minimization of movement within the hospital. On the other hand, the internal environmental factors such as atmospheric conditions (pressure, temperature, relative humidity, odour and particle pollution), sound, light and fire protection produce constraining effects on the arrangement of spaces since certain spaces cannot be placed adjacent to other spaces because of different requirements in environmental conditions. The consideration of logistics is important at all levels of the hospital system.

For example, the travel patterns between objects in a zone or those between zones in a room are frequently equally important for devising an effective design. On the other hand, the adjacency desirability matrix based upon environmental conditions will not be important for organization of functional elements below the room level since a room is the lowest level that can provide a physical barrier to contain desirable environmental conditions. Hence, the organization of functions for a new hospital can be carried out through an interactive process, starting from the functional elements at the lowest level that is regarded as stable by the designer, and moving step by step up to the top level of the hierarchy. Due to the strong correlation between functions and the physical spaces in which they are performed, the arrangement of physical spaces for accommodating the functions will also follow the same iterative process. Once a satisfactory spatial arrangement is achieved, the hospital design is completed by the selection of suitable building components which complement the spatial arrangement.

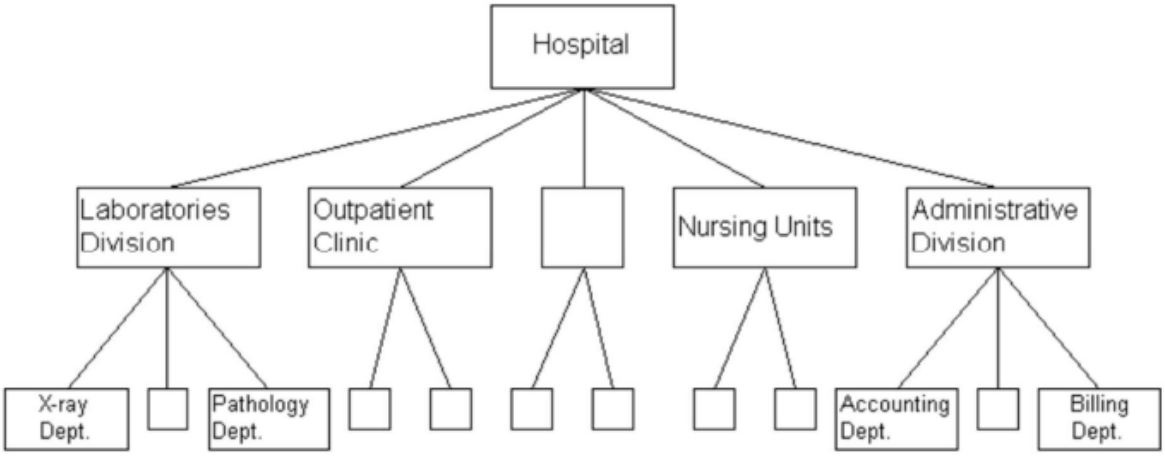


Figure 3-6: A Model for Top-Down Design of a Hospital

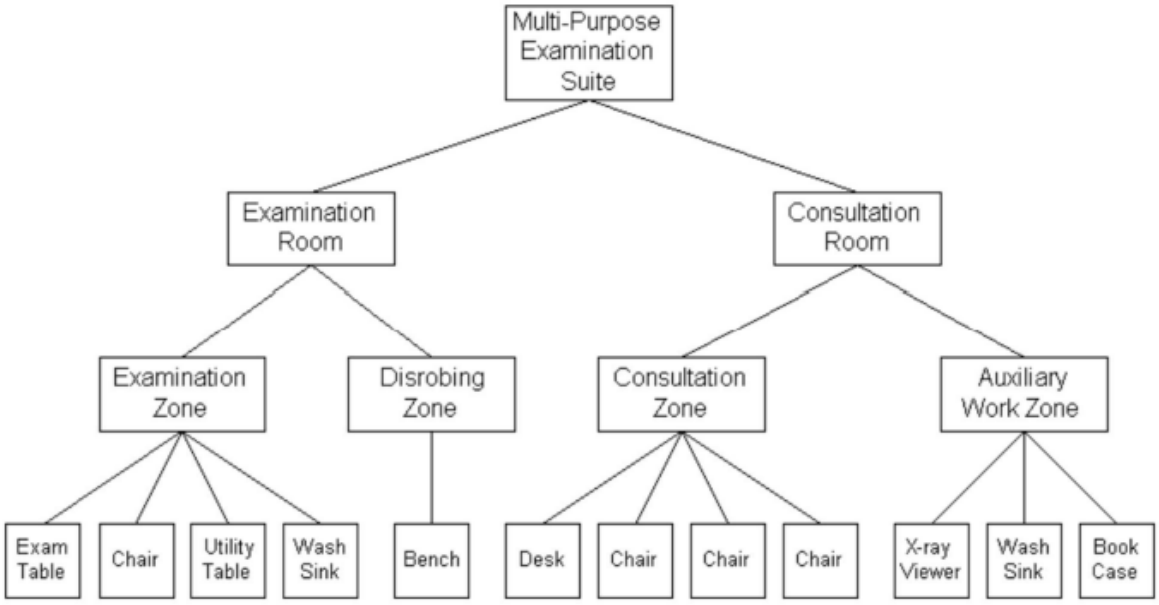


Figure 3-7: A Model for Bottom-up design of an Examination Suite