



NTNU – Trondheim
Norwegian University of
Science and Technology

A Modular Design Methodology for OSV Accommodation Areas

Zahid A.S.M. Kawser

Marine Technology

Submission date: June 2012

Supervisor: Stein Ove Erikstad, IMT

Co-supervisor: Veronica Rode, STX OSV Design

Norwegian University of Science and Technology
Department of Marine Technology

Master Thesis in Marine Systems

Design Stud. techn. Zahid Kawser

“A Modular Design Methodology for OSV Accommodation Areas” Spring 2012

Background

In most offshore support vessels today, the accommodation area is to a large extent designed from scratch towards each customer’s individual requirements, and within the available spaces given by the individual hull form.

STX OSV is currently investigating the possibility of developing pre-defined accommodation modules that can be re-used across different projects, and the corresponding templates and processes to support a cost efficient configuration of these modules towards the specific requirements of each individual project.

Overall aim and focus

The overall aim of the project is to develop and test a modular design methodology for the accommodation area of offshore support vessels.

Scope and main activities

The candidate should presumably cover the following main points:

- 1. Provide an in-depth, critical review of existing theory and methodology related to modularization, focusing on core challenges in developing a design process based on these principles.*
- 2. Apply different modular methodologies (such as Modular function deployment, Modularity matrix etc.) to develop a modularization strategy for the different functions being part of the vessel accommodation.*
- 3. Further develop the framework outlined in the MSc project to determine module size, type and mix, as well as identifying effective arrangements. Discuss both the designer’s role in this process, as well as the applicability of optimization models for decision support in each step of the process.*

4. *Describe STX's current approach of reengineering the accommodation design process, and compare this with the method developed in (2)-(3).*
5. *Develop a mock-up of a tool to support the process.*
6. *To the extent possible, exemplify and verify the proposed approach with a simple case study for an OSV design*


Modus operandi

At NTNU, Professor Stein Ove Erikstad will be the responsible advisor.

The MSc project is within the topic area of the KMB project SHIP-4C, and is thus eligible for travelling grants from this project.

The candidate will collaborate with STX OSV. The contact person at STX OSV will be Veronica Rode. Any insight into sensitive business information, or access to confidential data, resulting from the collaboration with STX OSV, should not be included in the project report without the explicit permission from STX OSV.

The work shall follow the guidelines given by NTNU for the MSc Thesis work.



Stein Ove Erikstad
Professor/Responsible Advisor

Nomenclature

AHP	Analytical Hierarchy Process
AHTS	Anchor Handling & Tug Supply
BFS	Breadth First Search
DBB	Design Building Block
DSF	Decision Support Framework
DSS	Decision Support System
FIFO	First In First Out
GA	General Arrangement
GUI	Graphical User Interface
HVAC	Heating, Ventilation and Air Conditioning
MFD TM	Modular Function Deployment
MIM	Module Indication Matrix
MOM	Measures of Merit
OPV	Offshore Petrol Vessel
OSCV	Offshore Support & Construction Vessel
OSV	Offshore Support Vessel
PSV	Platform Supply Vessel
QFD	Quality Function Deployment
SBSD	System Based Ship Design
WC	Water Closet

Preface

This dissertation is the outcome of a research work conducted in the 4th semester of my master studies at the Department of Marine Technology, Norwegian University of Science and Technology (NTNU). This research work is built upon my pre-master thesis titled “Modularization and configuration based design of offshore support vessel accommodation areas”. It also relies on the report of my summer internship in NTNU and STX OSV Design AS in August 2011. The dissertation along with the pre-master thesis and the summer internship is part of the ‘Ship-4C’ project run by the Department of Marine Technology. STX OSV Design AS is a collaborating partner in all of these research works.

STX OSV has been at work to develop a modular construction policy for the accommodation blocks of its offshore vessels. This thesis looks into different methodologies for modular production and identifies the key spaces that should be modularized for a modern production strategy. A decision support system for the designers of accommodation of OSVs is another major outcome of this research work. Although the primary goal was to complement STX’s modular concept; the course of action was slightly modified along the way. Following the suggestion of my supervisor, later the goal was set to work independently and come up with some fresh ideas for modular OSV accommodation. Rather than expanding the current STX approach of modular construction, this research work has taken a different route starting from the central theories of modularization and then building up from there to a decision support system for assisting the designers of OSV.

I would like to thank my research supervisor Professor Stein Ove Erikstad for his outstanding support throughout the research work. I am also grateful to him for giving me the opportunity to work in the summer internship and subsequent Ship4C project. I would also like to express my gratitude to Veronica Rode, Industrial Designer at STX OSV Design AS for her valuable insights in accommodation block design. Special thanks go to my friend Ayesha Tasnim for helping me with the decision support framework and corresponding programming for the real case study.

Trondheim, 10th June 2012

Summary

Offshore shipbuilding industry is known for its highly customized products, which are in most cases tailor-made for specific missions. Being a traditionally conservative industry, it follows conventional design practices. Because of the urge to quickly respond to changing market situations and the need for a structured method to reuse design knowledge across different projects, modern design methodologies like modularization are of considerable interest in this industry. To implement modular thinking, it has to overcome many challenges such as non-functional engineering description of design, scaling, clustering, logistical issues, structural complexities etc.

Different modular methodologies can be adopted to establish a modular product platform. Modular Function Deployment (MFDTM) is a popular theory of modularization with five distinct steps to identify the objects that should be modularized and optimize them for the whole manufacturing system. Application of MFDTM method reveals that among various spaces inside the accommodation of an OSV, crew single, crew double and officer's cabins are the most appropriate ones to be modularized.

Based on the output of MFDTM method and the vessel database from System Based Ship Design (SBSD) approach for OSVs, a decision support framework has been developed to assist the naval architect of the vessel to design the accommodation block. Standardized templates are used for arrangement of the modules. The spaces in the accommodation are categorized into three module classes. The DSF can calculate the optimum size, type and location of the modules according to design requirements and system constraints. The vessel database is used here to determine the required area for the spaces that are not standardized. The DSF is demonstrated by an illustrative example and later by a simplified real case study. A mock up of a user interface has also been developed to give an idea of user interaction with a computer tool based on the DSF. For the real case study, a small program has been developed containing some 1200+ lines of code, which can process simplified accommodation design cases.

To build an effective DSS for designing the accommodation block of OSVs, standardization of spaces by modularization must be done with enough room for flexibility of design by designer intervention. This DSF exhibits the underlying logics and structure of such DSS, which can be very useful to the designers upon further development.

Table of Contents

Nomenclature	iii
Preface	iv
Summary	v
List of figures	ix
List of Tables	x
1. Introduction.....	1
2. Modularization	3
2.1 Types of modularization.....	4
3. Modular design methodologies.....	6
3.1 Pahl and Beitz method.....	6
3.2 Matrix based methods	7
3.3 Functional flow heuristics.....	8
3.4 Modularity matrix method.....	10
3.5 Modular design in view of assembly complexity	11
3.6 Modular functional deployment.....	12
3.7 QFD-based modular product design.....	14
3.8 Relevant researches on modular shipbuilding	15
3.8.1 <i>Design building block</i>	15
3.8.2 <i>Optimization based space allocation approach</i>	17
3.8.3 <i>Intelligent ship arrangements</i>	19
4. Core challenges in developing a design process for OSV	20
4.1. Conventional thinking.....	20
4.2. Lack of structured approach.....	21
4.3. Lack of evident success in shipbuilding	21
4.4. Product platform and mix	22
4.5. Mode of production.....	23
4.6. Structural issues.....	23
4.7. Limitations on maximum module size.....	25
4.8. Concurrent engineering	25
4.9. Scaling issues	26
4.10. Different arrangement rationale	27
4.11. Limitations on crew module arrangement	28
4.12. Clustering.....	28
4.13. Ship: a deeply integrated system.....	29

5. Modularization strategy.....	30
5.1 Background	30
5.2 Modular Function Deployment (MFD™).....	30
5.2.1. Define customer requirements.....	32
5.2.2. Select technical solutions.....	34
5.2.3. Generate module concept.....	34
5.2.4. Evaluate module concept.....	37
5.2.5. Optimize modules.....	38
6. Decision support system	39
6.1 Module hierarchy.....	40
6.2 Base structure	41
6.3 Required input	42
6.4 Templates.....	43
6.5 Major assumptions.....	44
6.6 Summarized flow	45
6.7 DSS: The steps	46
6.7.1 User input phase.....	47
6.7.2 DSS initialization: Grid system for the decks	47
6.7.3 Template selection.....	50
6.7.4 Module specification.....	53
6.7.5 Higher ranked officer's module placement.....	54
6.7.6 Determination of the number of single/double crew modules	57
6.7.7 Common and fixed module placement.....	59
6.7.8 Scaling.....	61
6.8 Relevant theories	63
6.8.1. Bi partite matching:.....	63
6.8.2. Breadth first search (BFS).....	63
6.9 An illustrative example.....	65
User requirements.....	65
Modules.....	65
Graphical template.....	65
Finding number of crew modules.....	69
Fixed and common modules placement	70
7. Mock up of a user interface based on the DSS.....	72
7.1 Input phase.....	73
7.1.1. Selection of vessel type.....	73
7.1.2. Selection of class and notations.....	74
7.1.4. Customer requirements.....	76
7.1.5. Specification for fixed modules	77

7.1.6. <i>Specification for common modules</i>	77
7.1.7. <i>Designer specification for templates</i>	79
7.1.8. <i>Module specification according to the database and further adjustment</i>	80
7.2 Processing phase.....	81
7.3 Output phase.....	83
8. Real case study	85
Templates	86
Results.....	87
9. General discussion	90
9.1 DSS – the rationale	90
9.2 Comparison with similar approaches	92
9.2.1. <i>Van Oers approach</i>	92
9.2.2. <i>Design building block approach</i>	93
9.3 DSS – Known issues	94
9.4 Further development	97
10. Conclusion	98
11. Bibliography	99
12. Appendix	103
A. System Based Ship Design (SBSD)	103
B. Code for the real case study.....	105

List of figures

Figure 1: Different types of modularity [Ulrich, 2003]	4
Figure 2: An example of a clustered interaction matrix.....	8
Figure 3: Dominant flow heuristics	9
Figure 4: Flow branching heuristics applied to a generic structure.....	9
Figure 5: Modularity matrix [a. interaction matrix, b. transformed modularity matrix].....	11
Figure 6: Design building block approach applied to surface ship design [Andrews, 2006]	16
Figure 7: Design for production study to move the main machinery plant aft in an offshore support vessel to facilitate modular construction.....	17
Figure 8: Positioning space with filled and empty cells.....	18
Figure 9: Feasible ship design using polygons.....	18
Figure 10: Ship inboard profile (zone-deck system).....	19
Figure 11: Arrangement of spaces.....	19
Figure 12: Design spiral.....	20
Figure 13: Unrealized potential of modularity (Kusiak, 2002).....	21
Figure 14: Structural elements in cabin area	24
Figure 15: Modules and ship structure	24
Figure 16: Sequential Development Method vs. Integrated Development Method.....	26
Figure 17: Scaling possibilities.....	27
Figure 18: Module driver profile for accommodation block design.....	36
Figure 19: A possible arrangement template for an officer's deck.....	43
Figure 20: A possible arrangement template for a crew deck.....	44
Figure 21: The DSS.....	45
Figure 22: User input phase.....	47
Figure 23: DSS initialization	50
Figure 24: Template selection phase	51
Figure 25: Template suggestion	52
Figure 26: Template.....	53
Figure 27: Module specification.....	54
Figure 28: Flow chart for placing officer class modules	56
Figure 29: Determination of mix and placement of crew modules	58
Figure 30: Flow chart for placement of CM and FM	60
Figure 31: Scaling process.....	62
Figure 32: Bi-Partite matching	63
Figure 33: BFS method.....	64
Figure 34: Checking reachability by BFS method	64
Figure 35: Selected template for module placement.....	66
Figure 36: The deck consisting of sub grids.....	66
Figure 37: Cumulative sum grid	67
Figure 38: Grid/Cumulative sum grid after placing the staircases.....	68
Figure 39: Crew modules are being placed.....	69
Figure 40: Cumulative sum grid after placing the FMs.....	70
Figure 41: Possible module placement	71
Figure 42: User interface.....	72

<i>Figure 43: Selection of vessel type.....</i>	<i>73</i>
<i>Figure 44: Selection of class and notations.....</i>	<i>74</i>
<i>Figure 45: Vessel specification.....</i>	<i>75</i>
<i>Figure 46: Customer requirements on crew.....</i>	<i>76</i>
<i>Figure 47: Specification for FM.....</i>	<i>77</i>
<i>Figure 48: Specification for common modules.....</i>	<i>78</i>
<i>Figure 49: Template suggestion by the system.....</i>	<i>79</i>
<i>Figure 50: Template selection by the designer.....</i>	<i>80</i>
<i>Figure 51: CM specification by the system.....</i>	<i>81</i>
<i>Figure 52: Vessel database (sample).....</i>	<i>82</i>
<i>Figure 53: Processing window.....</i>	<i>82</i>
<i>Figure 55: System output.....</i>	<i>84</i>
<i>Figure 56: Input file for case study.....</i>	<i>86</i>
<i>Figure 57: Template for arrangement for the uppermost deck (Deck-D).....</i>	<i>87</i>
<i>Figure 58: Template for arrangement for lower decks.....</i>	<i>87</i>
<i>Figure 59: Output Deck D (uppermost).....</i>	<i>88</i>
<i>Figure 60: Output Deck C.....</i>	<i>88</i>
<i>Figure 61: Output Deck B.....</i>	<i>89</i>
<i>Figure 62: Output Deck A (lowermost).....</i>	<i>89</i>
<i>Figure 63: Design rationale for the framework.....</i>	<i>90</i>
<i>Figure 64: 3D output from Van Oers approach compared with the output from DSS.....</i>	<i>96</i>

List of tables

<i>Table 1: Modularity drivers.....</i>	<i>14</i>
<i>Table 2: List of spaces in accommodation block.....</i>	<i>32</i>
<i>Table 3: QFD matrix for accommodation block.....</i>	<i>34</i>
<i>Table 4: Module Indication Matrix (MIM) for module candidates for accommodation block.....</i>	<i>35</i>
<i>Table 5: Module hierarchy.....</i>	<i>40</i>
<i>Table 6: Notations used in the decision support system.....</i>	<i>46</i>
<i>Table 7: Template information table.....</i>	<i>52</i>
<i>Table 8: Possible module dimensions.....</i>	<i>65</i>
<i>Table 9: Template grid specification.....</i>	<i>67</i>

1. Introduction

Offshore shipbuilding is a part of the broad arena of shipping industry which caters vessels specifically to the offshore oil and gas industry, offshore exploration and seismic study groups, offshore wind mill etc. These industries need vessels that are highly customized and can meet special requirements on design and construction for serving in challenging conditions. The vessels produced by this industry include platform supply vessels (PSV), offshore support/construction vessels (OSCV), anchor handling and tug supply vessel (AHTS) and multipurpose vessels to name a few.

The accommodation block for offshore vessels normally holds a high standard, and the areas are to a large degree dependent on the number of persons onboard. The block is fitted at the stern or the bow of the vessel in order to have the much-needed open deck area/cargo area for offshore operations without any obstruction. The accommodation is therefore not adding much to the length of the vessel; the designer would rather add decks to make the necessary area [Vestbøstad, 2011]. Unlike cruise vessels, standardization of the accommodation areas of OSVs is not a straightforward task to accomplish. Because of the relatively small size of the block and highly variable design requirements depending on the mission, these accommodation blocks are custom-made in most of the cases.

Like other manufacturing industries, offshore shipbuilding industry has also been facing the challenge of transforming the vision from traditional design and production methods to agile manufacturing. A modern industry should be able to rapidly respond to all changes in the market environment. At the same time there is the growing urge to develop products that are of high quality but at a lower cost. Due to changing market situation and varying customer demands, modern shipbuilders are forced to reevaluate their design and production strategy and look for newer and more adaptable ones.

Different production strategies have been identified and examined for application in shipbuilding such as lean manufacturing, series production, modular production etc. The focus of this research work is modular production methodology. Modular production technique is nothing new to shipbuilding as naval and cruise shipbuilding industries have been using it for quite some time with varying success. Equipment and machinery suppliers for offshore shipbuilders such as Rolls-

Royce Marine and Wärtsilä have also been working hard to implement modular thinking in their product portfolios.

STX OSV is one of the prominent players in offshore shipbuilding involved in design and construction of a wide variety of offshore vessels. They are currently investigating the possibility of developing pre-defined accommodation modules that can be re-used across different projects, and the corresponding templates and processes to support a cost efficient configuration of these modules towards the specific requirements of each individual project.

In this paper, an in-depth look will be exerted into different modular methodologies. Chapter 2 will introduce the theme of modularization with definitions and types from various perspectives. Chapter 3 contains detailed discussions on several modular methodologies relevant to this research work. It will also summarize some modern approaches of modular thinking that are similar to this research. Based on available literature and discussions with designers of offshore vessels, the major hurdles that have to be overcome to successfully adopt modular product platform will be discussed in Chapter 4.

In chapter 5, Modular Function Deployment (MFDTM) method will be applied to identify the spaces, which should be modularized for a fitting production policy. Next, a decision support framework will be developed to assist the designer of the vessel on designing the accommodation block of offshore vessels with pre-defined modules in Chapter 6. This framework will also be explained with an illustrative example for easier interpretation. Chapter 7 will present a mock-up of a user interface for the aforementioned DSS. In Chapter 8, a computer program based on the DSS will perform a real case study. The framework will be compared with similar approaches for modular shipbuilding and its strength and weaknesses will be outlined in chapter 9. This will be followed by a revision of the decision support system and possible ways to improve it for developing a full-fledged computer tool for assisting the designers. Based on the DSS and subsequent discussions, some conclusions will be drawn in Chapter 10.

2. Modularization

Modularization is a term that goes back several decades and has been defined in various contexts. The term modularity in products is used to describe the use of common units to create product variants. It arises from the division of a product (part) into independent components, thus allowing one to standardize components and to create a variety of products [Huang, 1999].

Modular products and reconfigurable processes are crucial to agile manufacturing [Kidd, 1994]. Modular approach promises the benefits of high volume production (that arises from producing standard modules) and at the same time, the ability to produce a wide variety of products that are customized for individual customers [Huang, 1999]. By dividing the product into several parts, sharing of risk and investment can also be streamlined between suppliers and manufacturers. According to Baldwin and Clark (2000), “modularization is a strategy for organizing complex products and process efficiently”.

From Ulrich and Tung (1991), modularity is viewed as,

- (i) similarity between the physical and the functional architecture, and
- (ii) minimization of incidental interactions between physical components.

Another definition dating back to 1968 is also important in this context; “Constructed of modules or unit packaging schemes, usually having all major dimensions in accordance with a prescribed series of dimensions; capable of being easily joined to or detached, as an entity, from other components, units, or next higher assemblies” [Booz-Allen, 1968]

A modern definition from Schilling is, “A general systems concept: it is a continuum describing the degree to which a system’s components can be separated and recombined.” [Schilling, 2000]

From these definitions, modularization can be thought of as, “the division of a large system into independent individual parts/components which can be recombined into multiple end products” [Kawser, 2011]

In modular construction, the products are itemized into certain product modules that, when combined will generate different end products as a whole. Nilles (2001) points out that a product module is characterized by the following properties:

- a product module is a subsystem with lower complexity than the overall system of which the module is a part.
- a module is a closed functional unit.
- a module is a spatially closed unit.
- a module has well-defined and obvious interfaces.

2.1 Types of modularization

Modularization can be of different types. Depending on the mapping between the functional and physical elements of the product, Ulrich classified modularization in these three types:

1. Slot modularity
2. Bus modularity
3. Sectional modularity

A brief description for each type is given below:

Slot modularity: each of the interfaces between the components is of a different type from the others, which makes it impossible to interchange the various components of the product (e.g. the radio module in a car). Hence, there is only one possibility to connect the modules with each other.

Bus modularity: It is characterized by a common bus to which physical components are connected via the same type of interface. A typical example is personal computers where various components are attached to the common bus. Non-electronic products can also be built around a bus-modular architecture. Track lighting, shelving systems with rails and adjustable roof racks for automobiles all embody a bus-modular architecture.

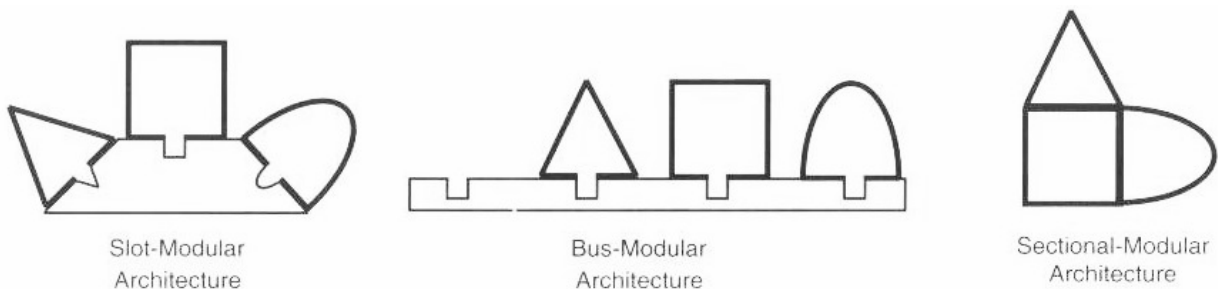


Figure 1: Different types of modularity [Ulrich, 2003]

Sectional modularity: The sectional architecture does not involve a single module to which the components can attach. The assembly is realized by connecting the components to each other via identical interfaces. This type is characterized by chained interconnection of modules. Many piping systems, office partitions, and kitchen furniture adhere to sectional-modular architecture [Ulrich et al, 2000].

Depending on the product and scope of construction, different type of modular architecture can be employed. In the development of a modular concept the objective is to find modules in which the contained technical solutions have similar properties regarding development, variety, processes, etc. [Kenger, 2006]. There is also another class of modularization, which basically shares features from the three aforementioned types. Mixed Modularity combines several standard components together through webs of modules rather than a simple chain, as with sectional modularity. The modules must be equipped with at least two complementary interfaces to create a new device [Kawser, 2011].

Again, Pahl and Beitz classified function modules into four categories: basic, auxiliary, adaptive and non-module [Pahl and Beitz, 1988].

- A basic module is a module implementing basic functions. The basic functions are not variable in principle and are fundamental to a product or system.
- An auxiliary module corresponds to auxiliary functions that are used in conjunction with the basic modules to create various products.
- An adaptive module is a module in which adaptive functions are implemented. Adaptive functions adapt a part or a system to other products or systems. Adaptive modules handle unpredictable constraints.
- A non-module implements customer-specific functions that do occur even in the most careful design development. Non-modules have to be designed individually for specific tasks to satisfy the customer needs [Huang, 1999].

3. Modular design methodologies

Research from different perspectives has been carried out to implement modular product architecture in overall product platform. Depending on the approach used, modular design methodologies fall into the following classes:

1. Pahl and Beitz method
2. Matrix based methods
3. Functional flow heuristics
4. Group technology based approach
5. Consideration of technology complexity
6. Modular functional deployment (MFD™)
7. Modularity matrix method
8. Quantitative functional modeling method
9. Product modularization for life cycle engineering
10. Developing modular products for testability
11. Modularity operation of systems based on maintenance consideration
12. QFD-based modular product design

[Nepal, 2005]

Not all of these approaches are relevant to shipbuilding industry. In this section, the methodologies that are important with regards to modular shipbuilding will be discussed in details.

3.1 Pahl and Beitz method

Pahl and Beitz refer to modular products as machines, assemblies, and components that fulfill various functions through the combination of distinct building blocks or modules. The overall design of the product is broken down into designs for separate functional modules. Each module can then be considered independently with the interactions between them being kept to a minimum [Pahl et al, 1988]. From a functional structure diagram, components of a product are selected to cluster based on their cost effectiveness in relation to the assembly cost of the entire product.

The essence of developing modular products can be summarized as follows:

1. Clarify the task: Generate specifications.
2. Establish a functional structure: Subdivide the main functions into a minimum number of similar and recurring sub-functions.
3. Determine the methodology to be used to implement the sub-functions. Also identify solution principles for implementation of the variant sub-functions.
4. Explore the feasibility between interfaces of modules and basic components (geometric, kinematics, and non-motion machine primitives).
5. Review the constraints.

The major advantage of this approach is the simplification of the subsequent design process for the individual modules. The major disadvantage of this method is that by reducing the scope for functional sharing, an increase in overall complexity of the product often follows. This can result in manufacturing problems, such as a higher parts count. Another shortcoming is that there is no clear optimization rules/techniques for module selection.

3.2 Matrix based methods

Pimmler and Eppinger, 1994 proposed an integration methodology for product design decompositions. The method involves three steps:

- 1) Decomposition of the system into elements,
- 2) Documentation of the interactions between the elements, and
- 3) Clustering the elements into architectural and team chunks.

This method helps to better understand the complex interactions of the constituting components of a product. By the help of the system integration matrix and clustered integration matrix, one can analyze given design decompositions [Pimmler et al, 1994].

Here, the researchers used a heuristic swapping algorithm facilitated by computer spreadsheet software and specialized macros for identifying the team chunks. The shortcomings of this

method are the need of specialized algorithm for different types of product design and a lack of focus into architectural issues that should be considered in the clustering process [Nepal, 2005].

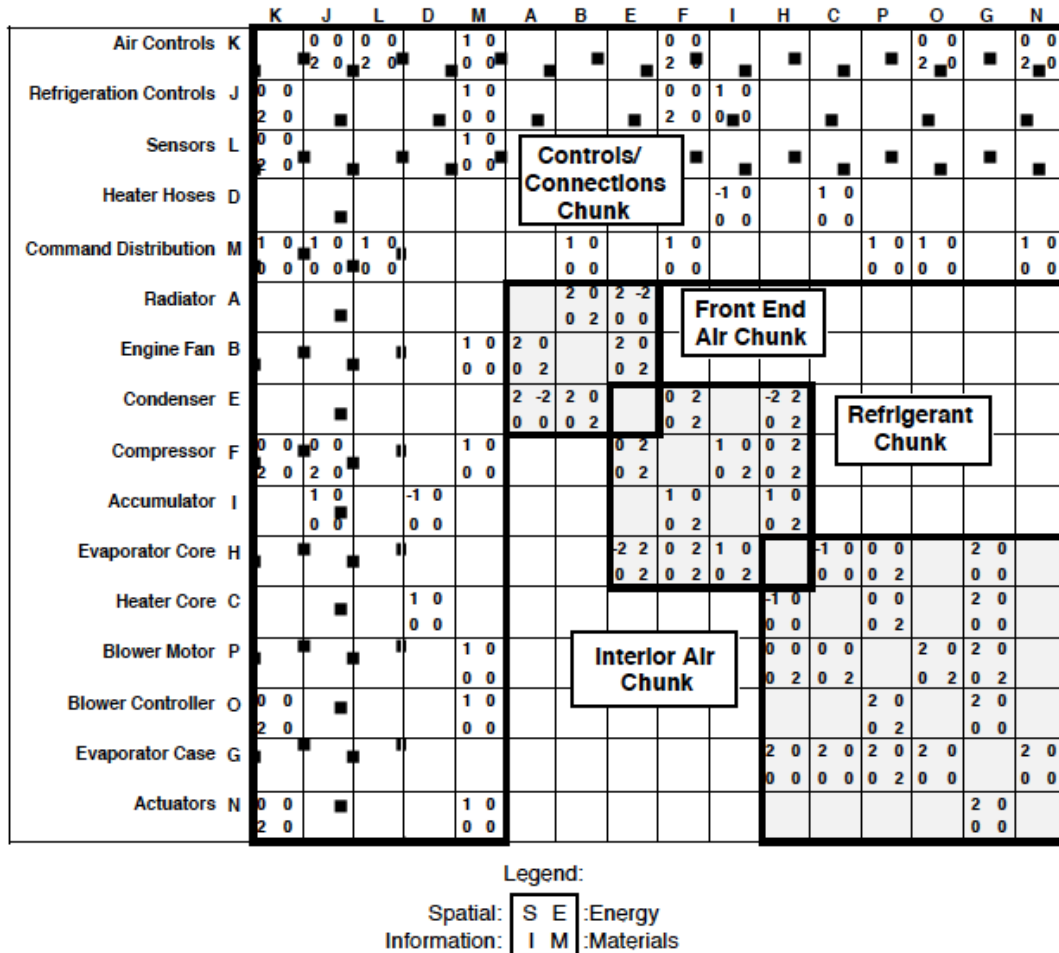


Figure 2: An example of a clustered interaction matrix

3.3 Functional flow heuristics

When considering a single product, Stone et al. (2000) identified a set of three heuristics that can be used to identify product modules on a function structure. The heuristic methods applied to modularize product function structures are divided into three types: dominant flow, branching flows, and conversion–transmission.

The dominant flow heuristic examines flows through a function structure, following flows until they either exit from the system or are transformed into another flow. The sub-functions through

which a flow can be traced, define a module. In other words, a set of sub-functions through which a flow passes, from initial entry or formation of the flow in the system, through final exit or conversion of the flow within the system, define a module.

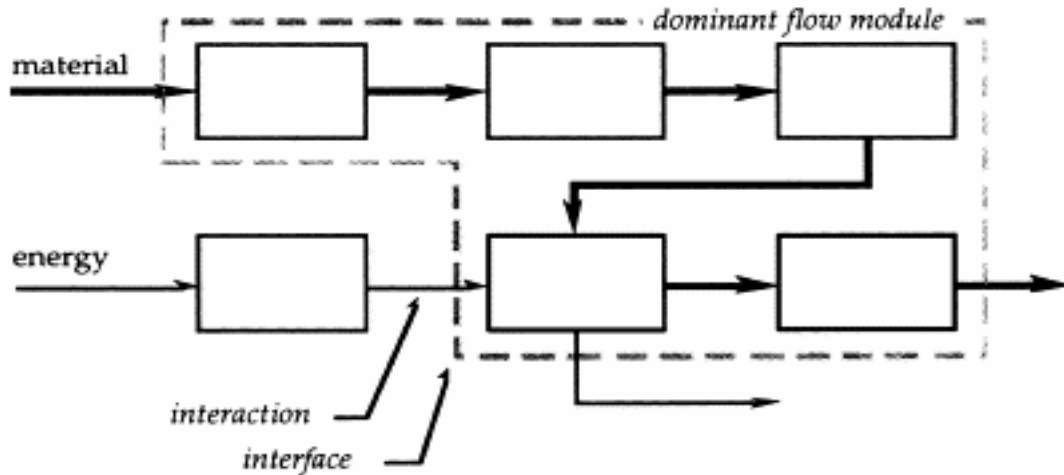


Figure 3: Dominant flow heuristics

The branching flow heuristic examines flows that branch into or converge from parallel function chains. Each branch of a flow can become a module. Each of these modules interfaces with the product through the point at which the flow branches or converges.

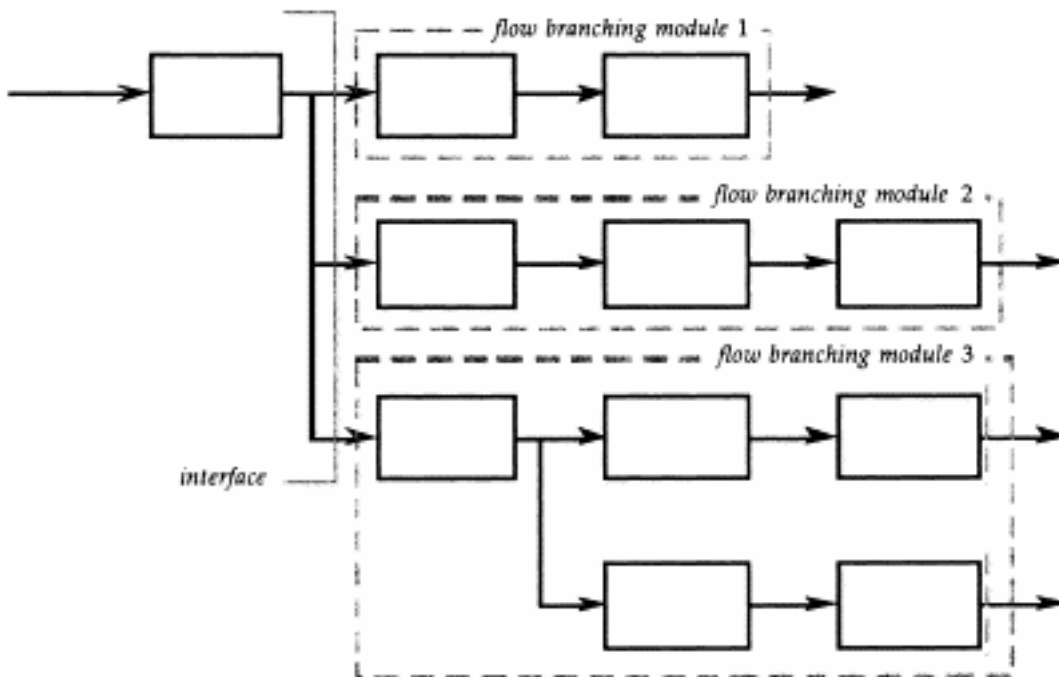


Figure 4: Flow branching heuristics applied to a generic structure

The conversion–transmission module examines flows, which are converted from one type of flow to another. A conversion–transmission module converts an energy or material into another form, and then transmits that new form of energy or material. In many instances, this conversion–transmission module is already housed as a module [Eggen, 2003].

Dahmus et. al. presented two additional heuristics to find common modules across products in a product family. They find shared functions across products, and unique functions that are found only in one product within the product family and separate them as modules [Dahmus et. al. 2001].

3.4 Modularity matrix method

An extension of Stone et al.'s (2000) functional flow heuristics is the modularity matrix method by Dahmus et al. (2001). They presented an approach to architecting a product family that shares inter-changeable modules. Rather than a fixed product platform upon which derivative products are created through substitution of add-on modules, the approach here permits the platform itself to be one of several possible options.

The architecture process begins with what underlying technologies should be utilized, and by establishing the limits of the product family that must share common modules. In the next step, function structures for each of the product concepts are developed. These individual product function structures are then combined into a large family function structure. The next step is to construct a modularity matrix.

The modularity matrix is a tool designed to aid the application of modularity rules both for products and portfolios. It lists the possible functions from a family function structure as rows in the matrix, then the possible products from that family as columns. Each matrix element contains a value that represents the required function specification level [Dahmus et al, 2001].

Once the family function and individual product structures are developed along with the modularity matrix, various architectures are developed by changing the target within the matrix according to modularity rules. There are, however, two aspects to this: developing the individual

product architecture for each product, and developing any shared modules within the portfolio architectures. First, individual architecture is developed by grouping the functions using the modularity rules on the function structure. These groupings are then displayed in the modularity matrix.

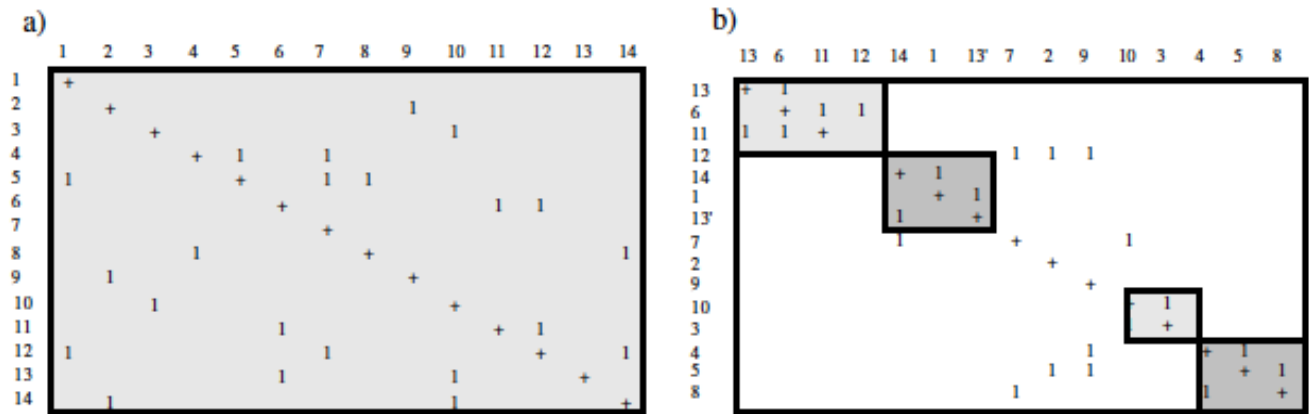


Figure 5: Modularity matrix [a. interaction matrix, b. transformed modularity matrix]

Next, the portfolio architecture is explored by identifying the shared functions across the portfolio. Such sharing can be found by examining similarities across columns for a single function in the modularity matrix. To establish shared modules, one has to determine whether different products have similar specification targets. One major shortcoming of this method is, apart from the similarity factor, no other performance metrics are considered here [Nepal, 2005].

3.5 Modular design in view of assembly complexity

The researchers, Tsai and Wang (1998), presented a methodology of modular-based design in the conceptual stage of systems to support concurrent engineering.

It consists of three steps:

1. The functions are classified into different types of modules according to the correlation in design by using fuzzy cluster identification.
2. The optimal module type is selected based on the considerations of the manufacture and assembly complexities of the system.
3. The design priority of functions within a module is scheduled by measuring the information content of functions.

While assessing the correlation of functions with each other in designing, four I/O parameters are used in order to check the similarity in I/O. The parameters used are geometric constraints, mechanical strength, energy flow, and signal flow. The functions are classified into different modules by using fuzzy clustering [Tsai et al, 1998].

Although the authors have presented the methodology to measure the level of manufacturing and assembly complexity and select the optimum modules, the paper does not consider other aspects of product architecture such as reliability, quality, and cost. Moreover, this methodology may be too complex for application in real cases.

3.6 Modular functional deployment

Modular function deployment (MFDTM) is a systematic method and procedure for company-supportive product modularization [Erixon and Ericsson, 1999]. The whole process consists of five steps.

1. From the analysis of competition and customer requirements, properties those must be possessed by the product are identified. Design requirements are derived from customer demands.
2. Functions that fulfill the demands and their corresponding technical solutions are identified. As there might be several technical solutions to fulfill a specific function, the method chooses only the most appropriate solutions based on certain evaluation criteria such as production goals, part number count, and future development potential.
3. In this step, the technical solutions selected in step two are analyzed regarding their reasons for forming modules using module drivers. This step is carried out with the help of a module indication matrix (MIM) wherein each technical solution is assessed against the module drivers. This step is the core of the MFDTM. Every technical solution is weighted on a scale of 1 to 9 where nine corresponds to a strong driver, three to a medium driver, and one to a weak driver in accordance with the importance of its respective reason for being module. The high weight or uniqueness in module driver patterns indicates that the technical solution under question has a complicated

requirement pattern and is likely to form a module by itself, or at least the basis for a module. On the other hand, few or low-weighted module drivers indicate that the technical solution under question might be easy to group together to form a module, provided there is a match in the module driver pattern or at least no contradictions.

4. Step four deals with the post-evaluation of the selected modules for the complete product based on the various performance metrics with respect to the interfaces between modules and the flexibility within the assortment. The suggested criteria for evaluating the effects of modularity in MFDTM include lead-time in development, development cost, development capacity, product cost, system costs, lead-time, quality, variant flexibility, service flexibility, service upgrading, and recyclability. Some of these criteria are just the rules that have to be followed during design; others are metrics, which can be measured and used for optimization of the modules.
5. Here, a specification is written for each module containing technical information, cost targets, planned development, descriptions of variants, and so on. The module specification constitutes the backbone of the product platform.

In essence, the process starts with quality function deployment (QFD) analysis to establish customer requirements and to identify important design requirements with a special emphasis on modularity. The functional requirements on the product are analyzed and technical solutions are selected. This is followed by systematic generation and selection of modular concepts in which the module indication matrix (MIM) is used to identify possible modules by examining the inter-relationships between "module drivers" and technical solutions. MIM also provides a mechanism for investigating opportunities of integrating multiple functions into single modules. The expected effects of the redesign can be estimated and an evaluation can be carried out for each modular concept. The MIM is then re-used to identify opportunities for further improvements to the single modules.

Ericsson and Ericsson have identified a number of driving forces for modularization within a product. These are called module drivers. These drivers cover the entire life cycle of a product and may be linked to different functions of the company.

<i>Module drivers</i>	
Product development and design	Carryover
	Technology evolution
	Planned product changes
Variance	Different specification
	Styling
Production	Common unit
	Process and/or organization
Quality	Separate testing
Purchase	Supplier availability
After sales	Service and maintenance
	Upgrading
	Recycling

Table 1: Modularity drivers

3.7 QFD-based modular product design

Kreng and Lee (2004) has consolidated the work of Ulrich and Eppinger (1995), Gu and Sosale (1999), and Erixon and Ericsson (1999) and extended it to a QFD-based modular design. Based on available literature, the researchers identified 14 modular drivers. Then Quality Function Deployment (QFD) is deployed to accomplish modular product design with two major phases. Phase one is the exploration of design requirements, which combines customer needs, company development strategies, and designers' preference to select proper modular drivers through competitive analysis. In phase two, modular product analysis and linear integer programming are used to establish final configuration [Kreng et al, 2004].

The 14 modular drivers are listed below:

1. Carryover
2. Technology evolution
3. Planned product changes
4. Standardization of common modules
5. Product variety
6. Customization

7. Flexibility in use
8. Product development management
9. Styling
10. Purchasing modularity components
11. Manufacturability refinement and quality assurance
12. Quick services and maintenance
13. Product upgrading
14. Recycling, reusing and disposal

They used a QFD framework to select the key modular drivers and to accomplish that they mapped the drivers with the specified design requirements. Analytical Hierarchy Process (AHP) is used to compare the different design requirements and assign weights to them. This would reveal the primary modular drivers. These modular drivers are ranked based on platform strategies and customer requirements. Next, they analyze the functional and physical relationship between each component pair. A linear integer-programming model is employed for the modularization process.

Although the authors demonstrated the application of this method on an electrical consumer product, this method may be too complex to apply in real cases.

3.8 Relevant researches on modular shipbuilding

3.8.1 Design building block

The building block design methodology developed by Andrews (1998) is quite relevant to develop modular offshore shipbuilding. Although this methodology was primarily intended only for naval vessels and submarines, the basis for this is established on functional attributes (float, move, infrastructure, fight etc.) that can be updated for specific sectors like offshore support vessels.

The UCL Design Building Block approach has been incorporated in the established PARAMARINE ship design system through a module entitled SURFCON. UCL's approach aimed to modernize the traditional ship design spiral sequence: initial sizing, parametric survey, layout, and performance analysis. Concurrent engineering principles supported a more integrated decision making process [Andrews, 2006].

The essence of the building block approach is first to select or define a series of design building blocks containing geometric and tentative ship size, which then are located as required within a prospective or speculative configurational space. A building block is identified by physical and functional attributes. According to Andrews, “The design building block should be thought of as a placeholder or folder in the design space containing all information needed to describe a particular function.” Overall weight and space balance and performance (e.g. stability, powering) of the design are assessed and the configuration is then manipulated until the designer is satisfied with both the configuration and the naval architectural balance [Andrews, 1997]. In three-dimensional space, the building blocks are controlled by drag and drop, and their properties dynamically update as their position is edited.

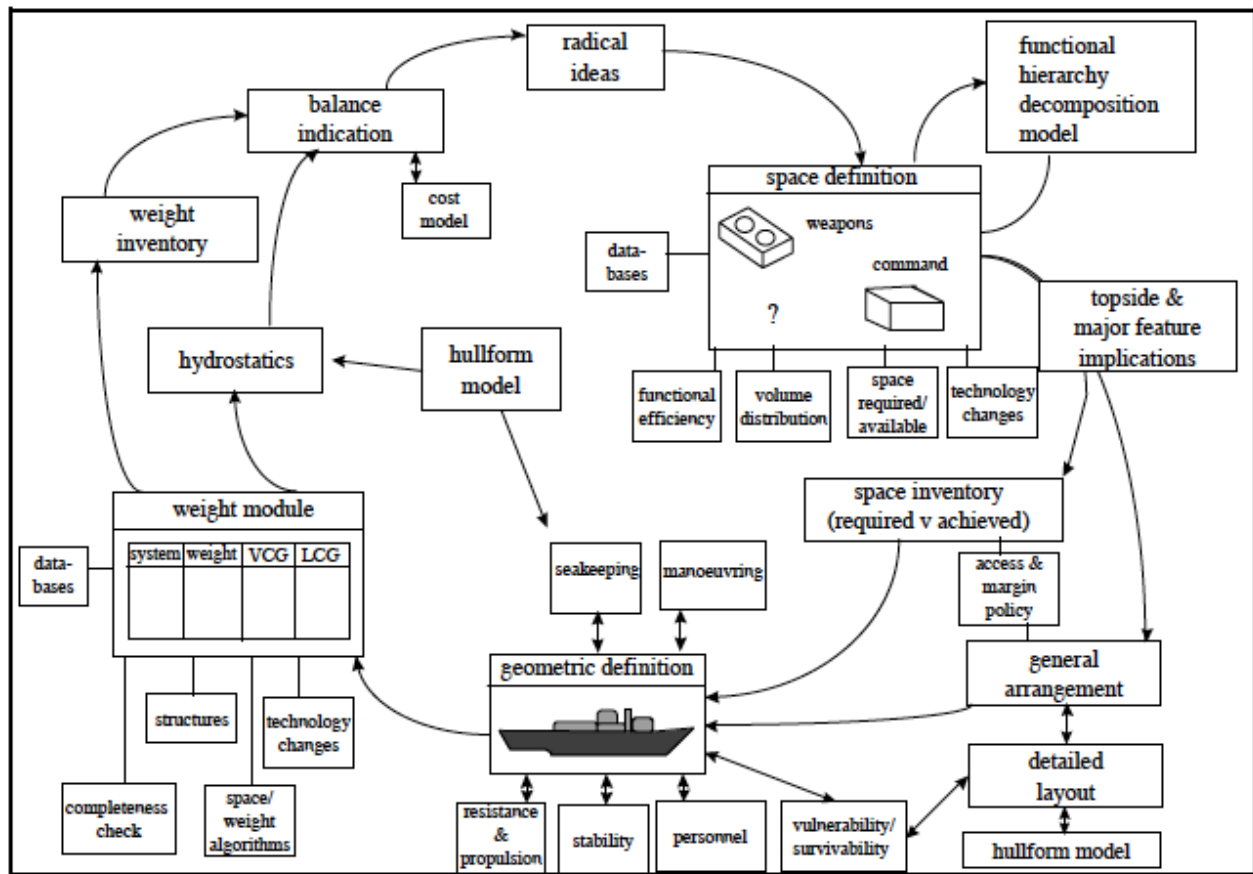


Figure 6: Design building block approach applied to surface ship design [Andrews, 2006]

Various case studies have been performed to assess the applicability of this approach for example novel designs of warships, SWATH, aircraft carrier etc. A relevant study has been undertaken on a North Sea offshore support vessel in which the main machinery space, previously under the forward deckhouse, has been moved to aft of the main drill supply tankage.

This change eliminated the twin shaft lines in the central region of the vessel, where they not only complicated the design of the main cargo space but also inhibited modular build off the slip, which would be a more efficient outfitting arrangement [Andrews, 2005].

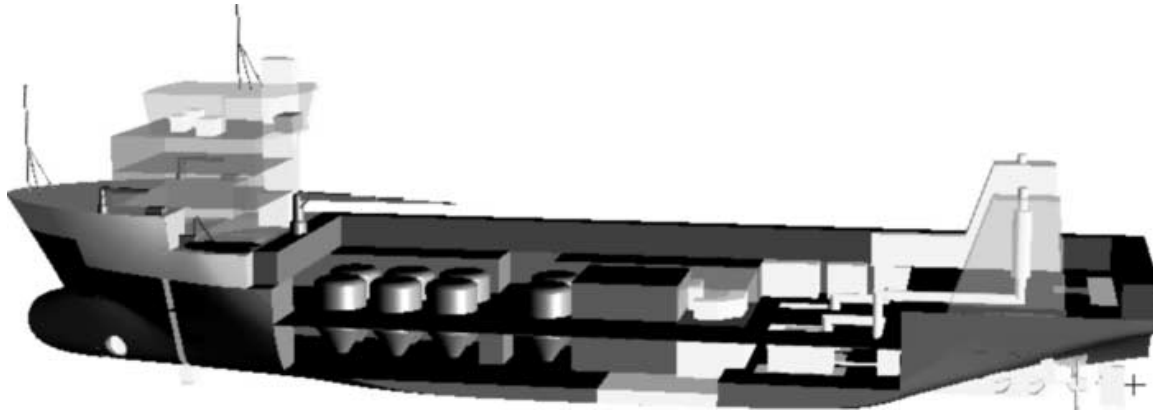


Figure 7: Design for production study to move the main machinery plant aft in an offshore support vessel to facilitate modular construction

This approach is focused on novel designs where the development period is large and the scope is broad. As Brathaug et al. 2008 suggested, the design space for offshore ships can be said to be a more closed design space. And in the case of accommodation block design for offshore vessels, the design scope is even more limited and the focus is given to efficient and timely solution for given customer requirements. But the emphasis on functions of the components can be seen as the first step towards modularization. The building blocks are distinguished based on functions, which can very well be treated as different modules for modular construction.

3.8.2 Optimization based space allocation approach

An optimization based space allocation method has been developed by Van Oers et al. 2007. The design is divided into objects, which have to be arranged. All objects have information regarding space demand, weight and center of gravity. These objects are then placed inside a total border. These borders can either be defined by interior or exterior demands.

To arrange the objects, the ship is presented as shown in Figure 7 with 0 for empty positions and non - zero for full cells. A deterministic approach is then used to position the objects. This is done by searching a vector for spots with the required number of zeros. If the object covers multiple cells in height, a vertical summation over rows covering the height of the object yields the required vector with cells.

Functional space allocation is done on a two-dimensional x-z plane inboard profile grid, with x positive towards the bow and z positive up from the keel. Spaces are all rectangular with fixed dimensions. Space height may exceed one deck. Location may also be fixed at the beginning of the algorithm. Available space is enclosed by a generic hull form that could be adjusted slightly to satisfy requirements; the upper envelope is unbounded [Brathaug, 2008].

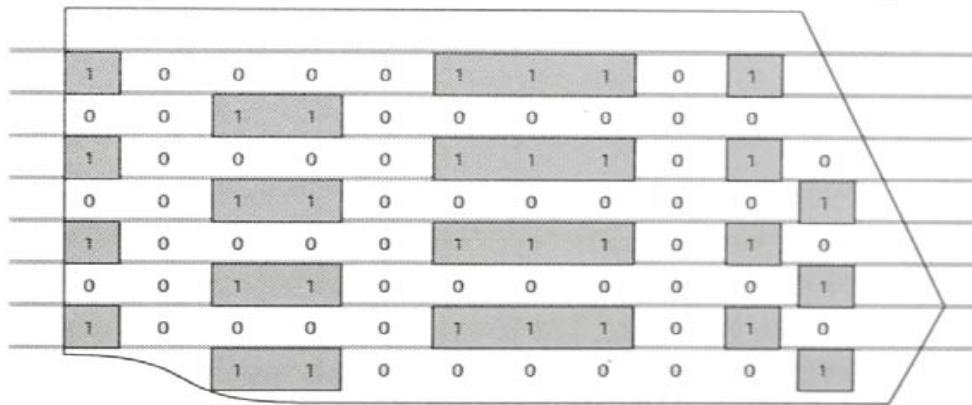


Figure 8: Positioning space with filled and empty cells

Four objective functions are used to optimize the designs based on genetic algorithm. These objective functions are: minimize total overlap, minimize center of gravity, minimize total void space and minimize onboard logistics.

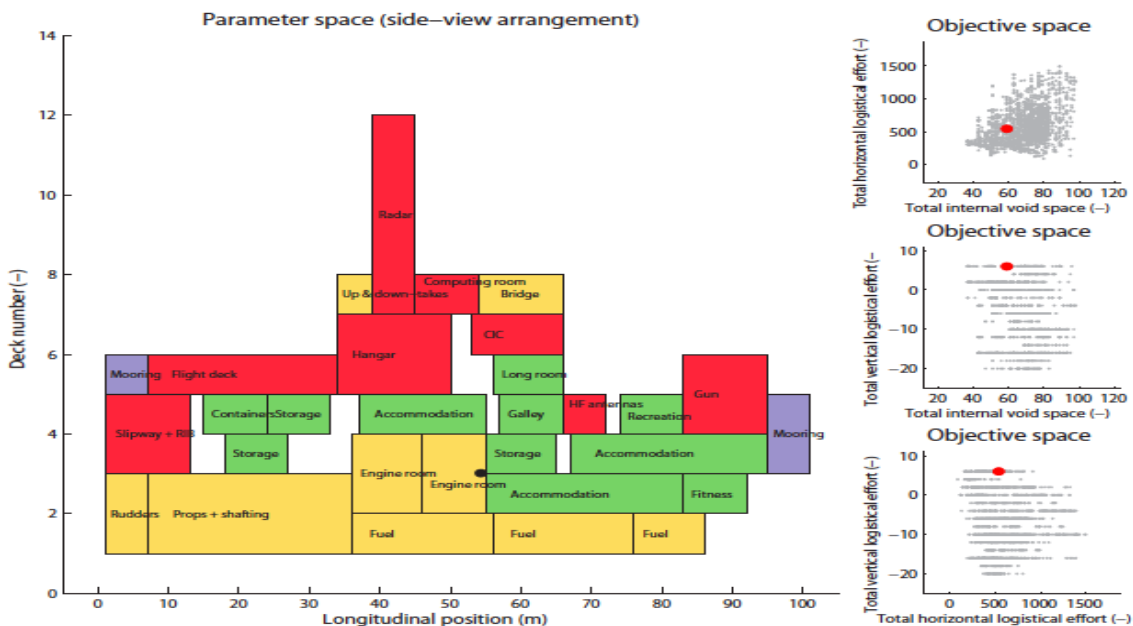


Figure 9: Feasible ship design using polygons

The primary criticism for this method is the representation of the design in 2D space where the objects are assumed to stretch through the whole breadth of the vessel.

3.8.3 Intelligent ship arrangements

Nick [2008] presented a research work that focused on arranging spaces such as galley, storage rooms, crew accommodation etc. onboard a frigate. This project was intended for naval application only. A fixed envelope, which is subdivided by decks and bulkheads, is created beforehand and arrangement takes place inside it. Some part of the ship that contains weapons and sensors as well as propulsion and auxiliary systems are excluded from the arrangement process.

A “zone-deck” system is used where spaces are arranged inside these in a coherent layout to ensure sufficient space is available and access to a network of passageways and staircases is provided. Fuzzy optimization method is employed as the search algorithm to find out the best design, which satisfies design and cost constraints.

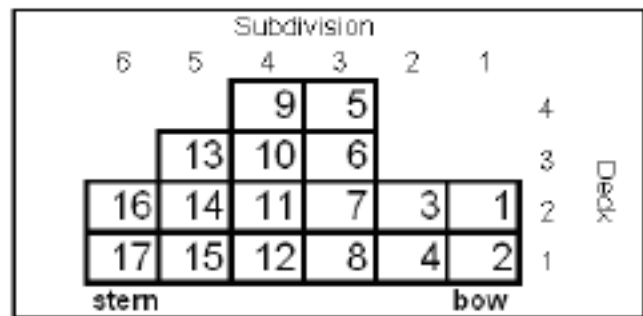


Figure 10: Ship inboard profile (zone-deck system)

The key criticism of this approach is the limited scope of application that excludes other types of vessels except naval vessels. The strict size and shape of envelope that cannot be varied for different cases is another disadvantage of it. The use of largely pre-defined passageways and staircases and the neglect of changes in center of gravity resulting from a different arrangement of spaces are also arguable characteristics of this approach [van Oers, 2011].

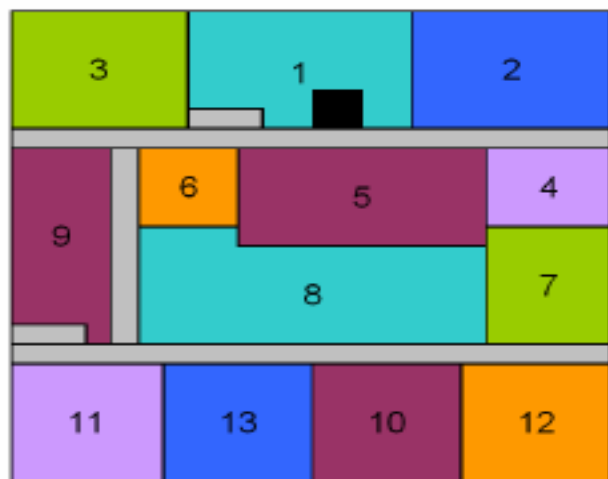


Figure 11: Arrangement of spaces (adjacencies and separation)

4. Core challenges in developing a design process for OSV

Designing accommodation areas in the modular way is a complex task. It has many issues to be resolved. Based on literature review and discussions with experienced designers in this field, a list of possible hurdles is summarized here.

4.1. Conventional thinking

Like other sectors of shipbuilding industry, offshore shipbuilding is a conventional one as well. Here, people tend to stick to the old norms and techniques of designing and building vessels. Since this approach has been working for them for ages, they do not want to change the proven formula and try a completely new approach. Another façade of the same problem is the practice of expressing the design requirements in engineering description rather than functional attributes. To adopt modular thinking, most of the approaches use some kind of functional structure where the constituent parts of the product are characterized by the functions performed. Here, various functions are identified and grouped to derive the necessary modules which can be functionally independent. In offshore shipbuilding this approach has not been used much. The traditional design spiral for satisfying the design requirements is the widely followed method in this industry. Despite clear unease about the descriptive adequacy (e.g. should it spiral inward or outward, the implied fixed sequence, how are feedback and new inputs or processes shown), naval architects have found it useful, if only to indicate the iterative nature of the design process and the consequent dependence of each downstream 'spoke' on its predecessors [Andrews et al., 1998].

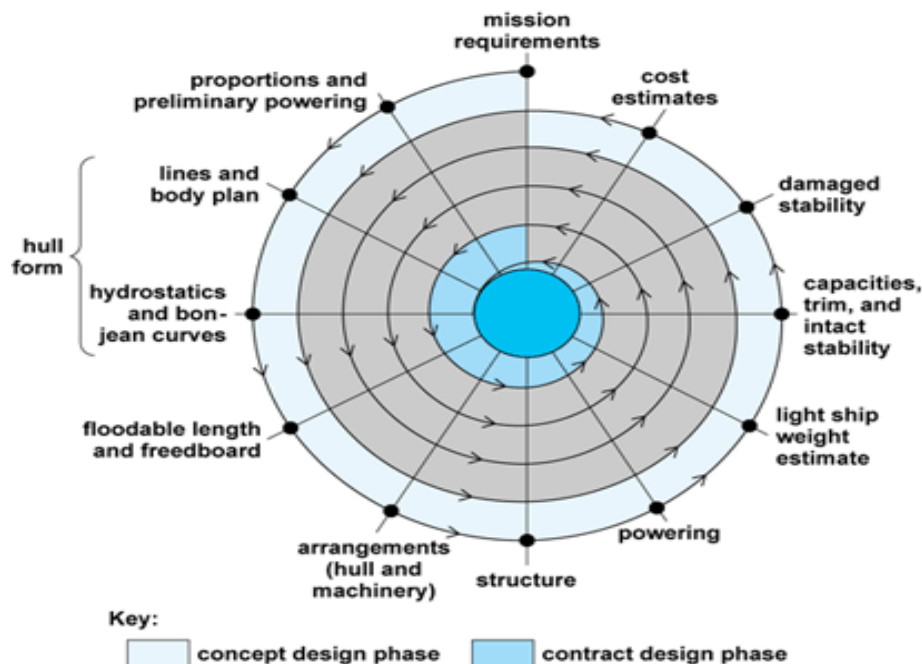


Figure 12: Ship design spiral

[Image: Larsson et al. Principles of yacht design, 3rd edition]

4.2. Lack of structured approach

There is currently no precise assessment of arrangements. They are typically generated by naval architects with years of experience dictating what works. Rules and criteria for these ad-hoc methods are documented, but they do not include all of the consideration needed for a strong arrangements solution. General arrangements remain very much an art [Nick, 2008]. Implementing a structured approach with modular product platform will radically change this long practiced way of thinking and substitute it with a modern and holistic process for designing vessels.

4.3. Lack of evident success in shipbuilding

One of the main criticisms of the modularity practice to date is the narrowness of the domain. According to Kusiak, this can be largely attributed to,

- Poor understanding of the modularity issue;
- Lack of theory and tools for the definition of modules from a broad perspective; and
- Some designers' skepticism of modularity's advantages because nobody has been able to demonstrate its benefits to them.

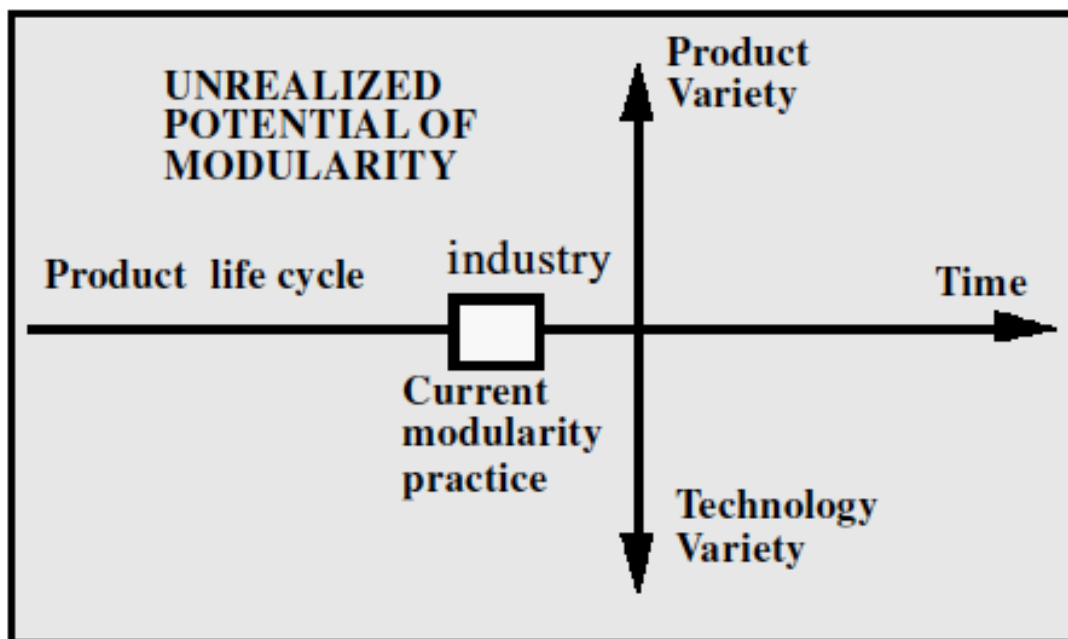


Figure 13: Unrealized potential of modularity (Kusiak, 2002)

The author has also demonstrated the potential of modularity by the above illustration. Here, the four white quadrants spreading across the product life-cycle axis and the product variety/technology axis represent the unrealized potential of modularity. To fully realize its potential, modularity must be redefined by enlarging the domain of products, encompassing different processes and technologies, incorporating the product life cycle cost, and increasing corporate products by improving the quality of the modules formed [Kusiak 2002].

As modular building is new to offshore shipbuilding, there are no generally accepted standards on module size, dimensions and build method. Offshore vessels are being built in different nations now a day such as Norway, Brazil, Vietnam, China, and Korea to name a few. It is difficult to reach specific standards when the design is being developed in a country whereas the actual vessel is being built in another.

4.4. Product platform and mix

Modular product platform consists of standardized modules that can be grouped together to derive different end products. Today's offshore shipbuilding industry follows the integral product platform approach. Introducing a modular platform will have some serious hurdles to overcome here.

Firstly, the level of standardization is an important issue. Since offshore shipbuilding is more of a customized practice than a rule-based standard affair, standardized components are a rarity in most cases. In order to implement modular architecture, whole product platform has to be reevaluated and products have to be identified which can be standardized based on system constraints and management decisions. Modular construction enables 'standardized diversity' by using different combinations of standard components [Stoll, 1986].

Secondly, introducing some standardized modules would be just part of the solution. Potential customers will always have some particular choices regarding the design in their mind and the product platform must have the flexibility to accommodate these extra specifications. The modules have to be routinely customized adding more custom features, components or finishes and other technological advancements. So, there are two opposing forces here; one to standardize everything and the other to limit the standardization and make room for

modifications according to customer choice. This is a very important issue when developing modular product platform and a fine balance has to be reached which will result in a sound modular architecture with enough room for designer modification.

4.5. Mode of production

Typically in the integral approach of vessel design and construction, the accommodation block including the cabins and other constituent parts are erected and assembled in the shipyard (on-site production). On the contrary, if modular building approach is followed, the cabins will be built in the dedicated factory/yard for module building (factory production). The workflow for production will be different according to the mode or place for production.

Factory production might require new material, equipment or system that are engineered specifically for production inside a factory/yard. Instead of the fixed position layout used on the construction site, modular manufacturers use product layouts - typically progressive assembly flow lines [Mullens, 2004]. Manpower requirements vary widely from module to module due to process randomness and product mix. For example, a module with no bathroom will require no cabinet installation, while a double crew module may have added requirements of office cabinets/cloth cabinets to install.

4.6. Structural issues

Modules are functionally independent entities that are placed onboard the vessel on designated areas to perform specific functions. To build the accommodation block in modular way, the vessel must have special structural arrangement that can support these modules. Many of the internal functions a module performs have to be connected to external support system or main line transmissions or piping systems for proper functioning. To facilitate these connections, steel structure of the vessel might have to be redesigned and special features be added.

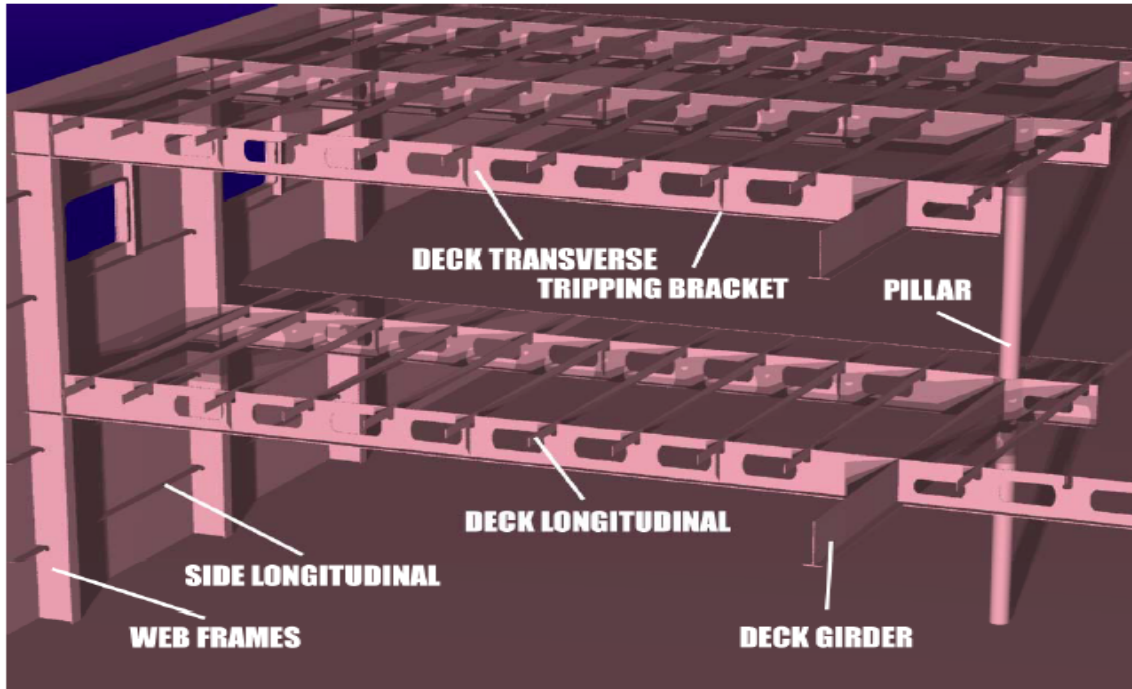


Figure 14: Structural elements in cabin area

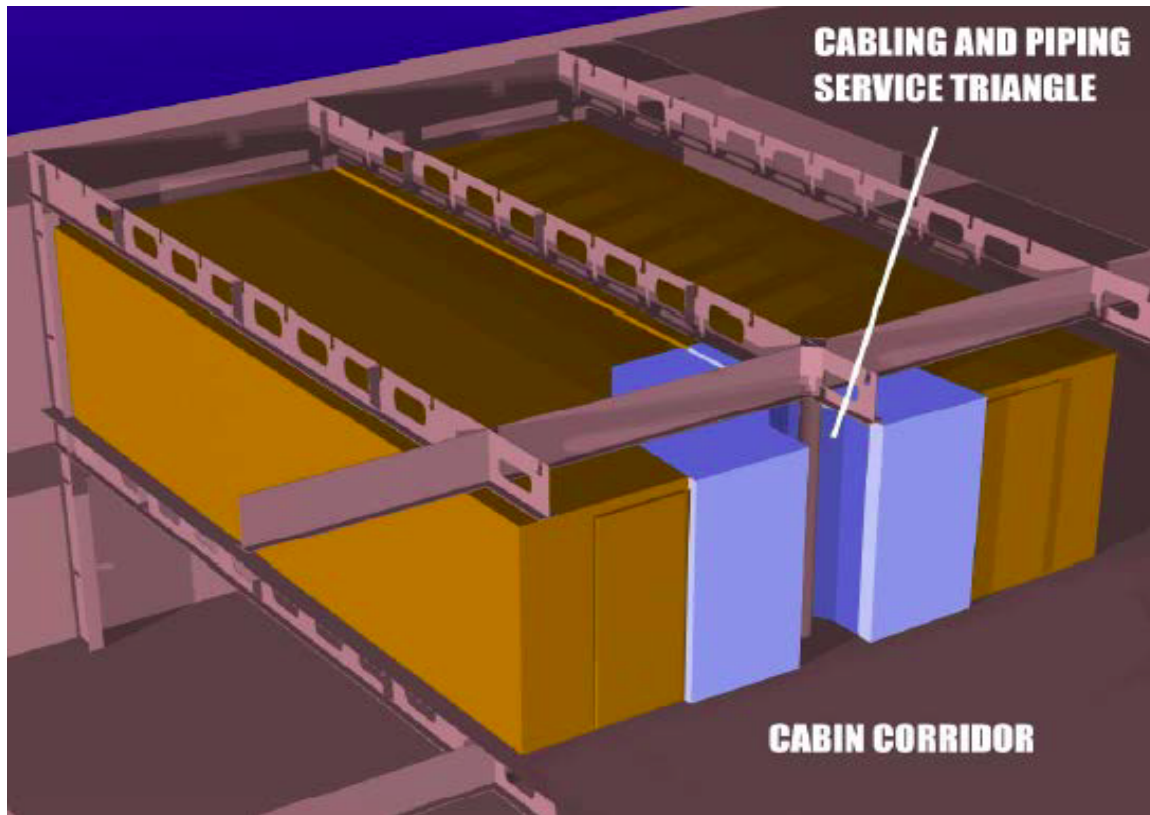


Figure 15: Modules and ship structure

[Image courtesy: Kai Levander, STX Europe]

In this picture, it can be noticed that the transverse girders are located along the edges of the modules and the cabling and piping arrangement is done in a special separate system. This kind of arrangement can demand special features on vessel structure.

4.7. Limitations on maximum module size

When a factory production system is followed for modular construction, the modules have to be transported from the factory to the assembly shipyard. Usually these are transported on trailers/lorries by roads. Available land connection is an important factor here to be taken into account when designing the modules. Getting finished modules to the assembly site without snagging power lines or having to negotiate too many hairpin curves are all crucial here to have an efficient transportation policy.

If the roads are narrow or there is strict limitation on cargo weights carried by the vehicles, then the maximum size of the modules have to be restricted. Typically the modules for cruise ships are dimensioned at 20 m² or below because of these considerations [Kai Levander]. The size of a module is often compromised due to several factors, e.g., the panel size, performance requirements, cost, or testability [Huang, 2000].

Size is important from testing purposes as well (testability). If the module is so large that it cannot be properly tested before assembly, then it is not very practical to have them constructed without performing necessary testing. Depending on the available testing facilities in the factory, the optimum size of the module has to be adjusted.

4.8. Concurrent engineering

Decomposition, standardization, and exchangeability are the attributes of a modular product. One of the vital advantages of adopting modular thinking is that if necessary, the modules can be updated or replaced when the vessel is in service. Life cycle of a module is a cardinal issue when designing the whole product portfolio.

Since every module is a separate independent entity, the development process for them can run in parallel; and hence comes the idea of concurrent engineering. Concurrent engineering is a work methodology based on the parallelization of tasks (i.e. performing tasks concurrently). It refers to an approach used in product development in which functions of design engineering,

manufacturing engineering and other functions are integrated to reduce the elapsed time required to bring a new product to the market.

It ensures that all elements of a product's life-cycle, from functionality, producibility, assembly, testability, maintenance issues, environmental impact and finally disposal and recycling are taken into careful consideration in the early design phases. This includes establishing user requirements, propagating early conceptual designs, running computational models, creating physical prototypes and eventually manufacturing the product [Kusiak, 1992].

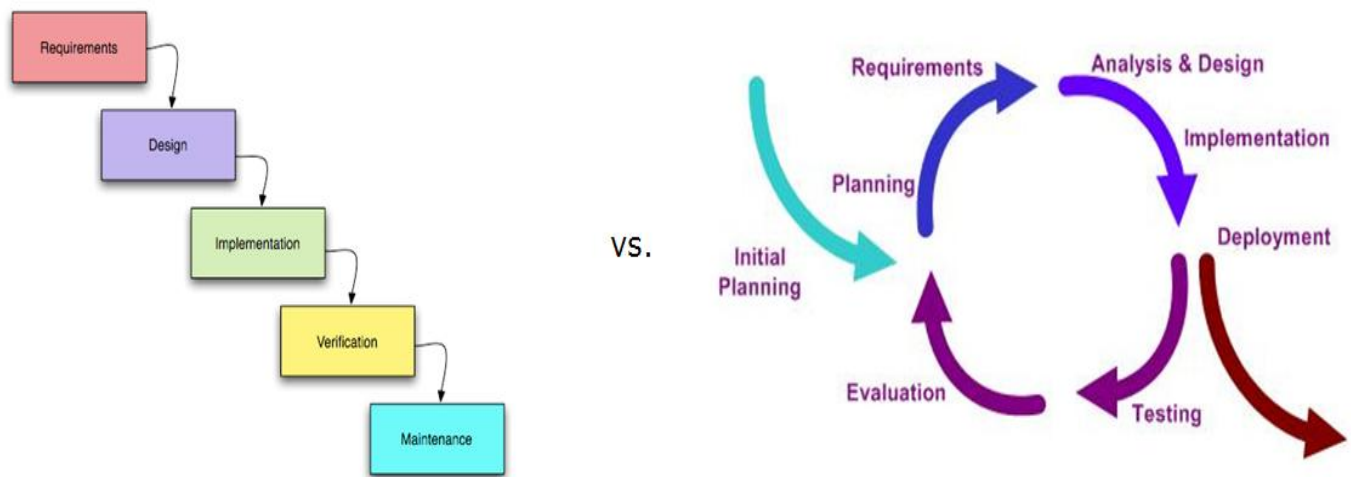


Figure 16: Sequential Development Method vs. Integrated Development Method

Since the offshore shipbuilding industry is a traditional one, this type of unconventional thinking could prove as difficult to implement. Concurrent engineering not only decreases the time for development of the modules, but it also ensures that quality, usability and other aspects of the product are getting the required attention in the development phase.

4.9. Scaling issues

Scaling is an essential factor when it comes to modularization. The modules have to be scaled depending on the ship type and space constraints. Scaling can be linear or ratio based where the ratio satisfies some length-width dimensional relation.

It can also be single directional or multi-directional. A multi directional scaling will increase/decrease the dimensions on both sides of the module (i.e. rectangular module). But a one directional scaling policy will scale the module in one specified direction only. This is

particularly important due to the curvature of the hull of the vessel. In most of the cases, the sideline of the vessel is curved and modules to be placed alongside this curved line will have to be scaled gradually to use the extra space, otherwise the space on sides will be of no use. This means the modules can be no longer rectangular; rather they would have one curved edge. Then again, the modules on the middle of the deck do not have this kind of restriction and can be scaled on both sides. So, modules on different areas may have to be scaled in different ways [Kawser, 2011].

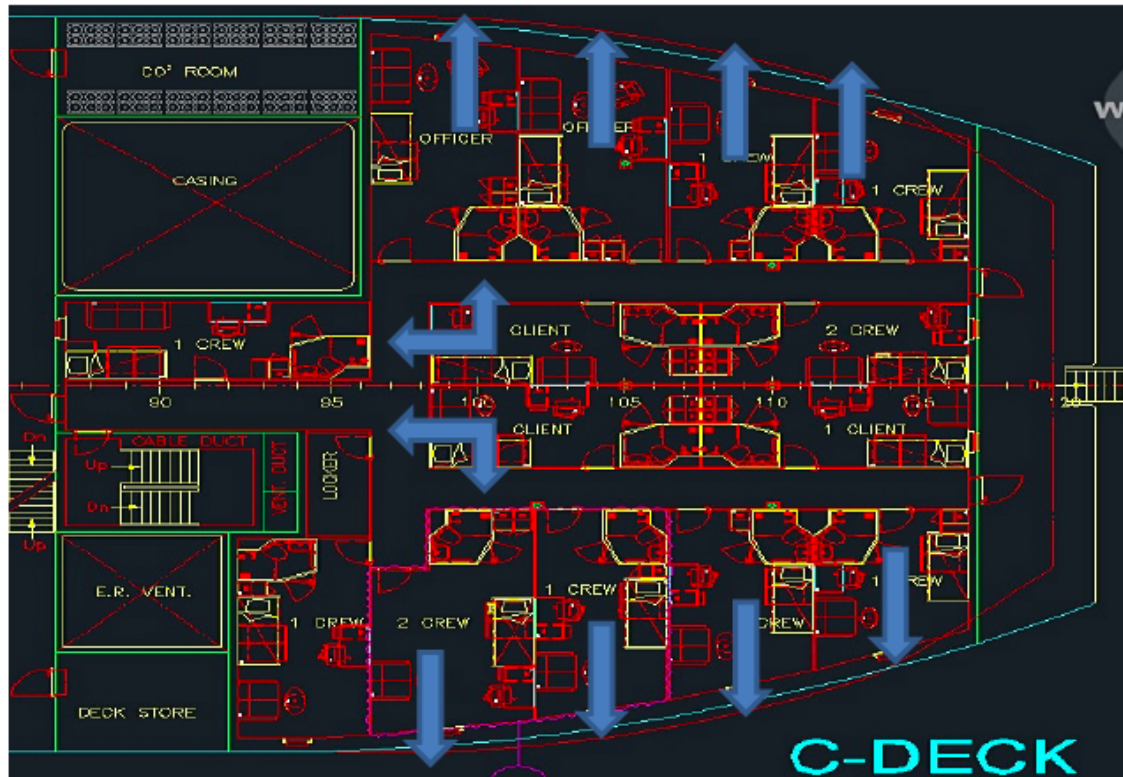


Figure 17: Scaling possibilities

4.10. Different arrangement rationale

Modules can be arranged based on different design perspectives. The choice on arrangement can pose restrictions to the overall design scope of the accommodation block. Some of them are discussed below:

- Standard arrangement: The floor plan for each floor can be standardized so that the crew and passengers have the sense of familiarity and ease when moving around the vessel performing their tasks. This is particularly important for larger construction vessels with crew of more than 100 people. Same setup on every deck level will make the crew feel comfortable offshore [Veronica Rode].

- Ease of evacuation: The arrangement can also be designed to aid quick evacuation in case of emergency like fire breakout etc. For this, the arrangement should be straightforward with ample space for corridors. The corridors should directly lead to exits without any bends or dead end.
- Space hierarchy: Usually higher ranked officers' cabins are placed on the topmost deck (one level below the bridge deck). Different types of modules can be developed for different ranks of crew and they can be grouped together according to the deck plan. If the vessel is large, then this method can reduce the complexity of dealing with so many modules by lowering the effective number of individual modules.

4.11. Limitations on crew module arrangement

Since crew comfort is one of the key issues when designing offshore vessels, the arrangement for their accommodation deserves more attention these days. Owners are demanding more and more features for the accommodation block where safety and comfort go hand in hand.

- Crew modules should be arranged such that each crewmember has own window view from his/her cabin [Veronica Rode]. This can only be possible if all the modules are placed along the sides of the vessel. But in many cases, offshore vessels have a large number of crew and some of the crew cabins have to be placed in the center of the vessel with no window view to outside. This is particularly true for larger construction vessels and anchor handlers. Solving this issue is difficult because of space limitation of vessels and the presence of other modules around the crew modules.
- The layout of decks is not standardized for offshore vessels. Fitting modules of the same fixed size on each deck is difficult when the layout differs on different deck levels. The varying shape of the superstructure also plays a role here in this regard.

4.12. Clustering

Modules of similar nature can be clustered to form a 'grand module block' that can be placed directly on designated areas of the deck. When designing a large vessel with lots of crew cabins, clustering can come particularly handy to limit the design variables to a minimum. This will make

the arrangement easier for the system. To implement this, the modules have to be designed such that they can be joined together in the factory or the yard before fitting them onboard.

4.13. Ship: a deeply integrated system

Unlike other industrial applications of modularization, modular shipbuilding has to overcome a rather herculean obstacle of dividing a system which is too integrated as a whole. An offshore vessel has numerous systems that are tightly integrated to one another so that the vessel performs as intended. To implement modularization, these systems have to be distinguished and separated based on their underlying functions. Here comes the problem of defining the boundaries of these systems and adequately separate them so that they can function individually without much interaction with adjacent systems [Erikstad, 2010].

For example, an offshore construction vessel can have an offshore crane module onboard to perform lifting and hauling operation offshore. This crane module can be thought of as a separate system that can be replaced or updated without interfering with other modules. But in reality, any alteration of this crane module would make huge impact to other modules. It would alter the total stability and structural balance of the vessel. It would necessitate changes in crane foundation module, power module, other connected support modules and most importantly the steel structure (hull and deck modules). Changing the parameters of a module on a vessel can cause ripple effect to the design and many other modules would have to be redesigned to accommodate this change.

5. Modularization strategy

5.1 Background

In order to develop a product platform with modular products, key product architecture decisions have to be made at the very early stages of product development. These decisions are linked to specific R&D issues, including the ease of the product change, division between internal and external development resources, ability to achieve certain types of technical product performance, and the way product development is managed and organized [Ulrich, 1995].

One-to-one mapping from functional elements in function structure to physical components is what makes modular architecture different from integral product architecture. Suitability of product architecture to different products depends upon many factors where type of product plays a vital role. There is no perfectly suitable architecture for products because design and manufacturing of a product can be done in different ways with varied scope of performance goals. According to Ulrich (1995), “In most cases the choice will not be a completely modular or completely integral architecture, but rather will be focused on which functional elements should be treated in a modular way and which should be treated in an integral way”.

To examine the suitability of these product platforms for OSV accommodation block design, the elements of design have to be identified and the function structures for these elements have to be mapped. It can be done in a lot of ways as described by the different modular design methodologies that have been listed and discussed in the previous chapter. Here, one of the major modular methodologies, the Modular Function Deployment (MFD™) method will be applied to offshore vessel’s accommodation block design.

5.2 Modular Function Deployment (MFD™)

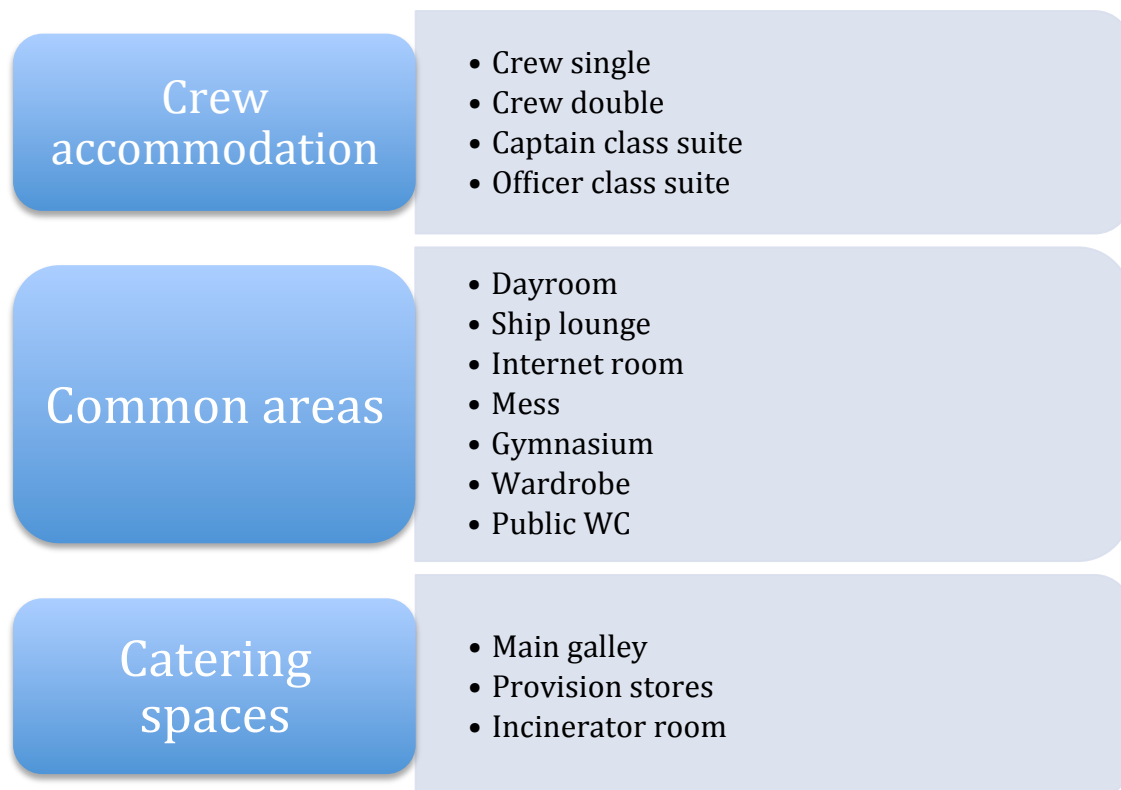
Modular Function Deployment developed by Ericsson and Erixon is based on quality function deployment (QFD) analogy for finding the optimal modular product design considering the company’s specific needs. Applications of MFD™ are found in a wide range of manufacturing companies in widely differing industries. It has been applied in modularization projects for cars, trucks, vacuum cleaners, staplers, grinding machines, servo drives, washing machines and many other products.

This is a 5-step procedure namely:

1. Define customer requirements
2. Select technical solutions
3. Generate module concept
4. Evaluate module concept
5. Optimize modules

These steps are already discussed in chapter 3. Here, these will be applied for determining the optimum modules for accommodation block design.

There can be different types of cabins/rooms in an offshore vessel depending on the type of service it is intended to perform. Hence, the accommodation block of a construction vessel will differ quite substantially than that of a platform supply vessel. The following table categorically shows the typical cabins/spaces of the accommodation block of an offshore vessel. The categorization here follows the System Based Ship Design (SBSD) approach developed by Kai Levander [Kawser, 2011].



Client accommodation	<ul style="list-style-type: none"> • Client/Owner's suite (Officer class) • Client/Owner's office
Ship service areas	<ul style="list-style-type: none"> • Conference rooms • Ship offices • ISPC office • Hospital
Hotel service	<ul style="list-style-type: none"> • Linen store • Ship laundry • Storage spaces • Cleaning lockers • Unpacking area
Construction related spaces in accommodation	<ul style="list-style-type: none"> • Technical room • Duty mess • Reception office • Various management offices • Project coordination offices

Table 2: List of spaces in accommodation block

The five steps of MFD™ system will be discussed with regards to the accommodation block.

5.2.1. Define customer requirements

In any design practice, it is vital to derive the appropriate design requirements from the customer needs. For accommodation block, typical customer (designer) requirements can be summarized as follows:

- High quality
- Easy to set up
- Flexible in terms of ship types
- Easy maintenance
- Adaptable to satisfy owner's requirements
- Transportable
- Long service life
- Low price

From these customer requirements, the corresponding product attributes or properties are derived. Product properties for the accommodation block might be:

- Size
- Weight
- Material
- Connector requirements (interfaces)
- Maintenance interval
- Comfort level
- Flexibility

The relations between customer demands and product properties are analyzed in the following QFD matrix. The relation is graded with a point system where a strong relation is equal to 9 points, a medium relation 3 points and a weak relation 1 point. The grade is then multiplied with the customer demand weight before it is summarized vertically. The analysis of different customer segments (in this case designers/firms with different business goals) will yield varying design requirements. This will consequently lead to differing target values for the corresponding product properties [Ericsson et al. 1999].

Product attributes										
Customer requirements	Weight	Modularization	Size	Weight	Material	Connector req.	Maintenance int.	Comfort level	Flexibility	Sum
High quality	5				3		1	3		7
Easy to setup	4		1	1					3	5
Flexible	5					9			9	18
Easy maintenance	3		3				9			12
Adaptable	5					3			1	4
Transportable	3	9	9	3					1	22
Long service life	3						3			3
Low price	2	3			1					4
Sum	X	33	40	13	17	60	41	15	65	

Table 3: QFD matrix for accommodation block

5.2.2. Select technical solutions

Here, the functional analysis of the system has to be carried out. The main objective is to map all functions, explain them and put them in their context. The MFDTM approach is suitable for mechanical systems whereas the accommodation block is a complex web of various systems such as electrical, mechanical, thermal and so on. Another important fact here is that the cabins/spaces are the utmost focus for modularization; not the internal systems themselves. Moreover, all these systems are largely separated from each other so drawing a functional mapping diagram is not going to be too helpful for this process.

Since the unit of design here can be identified as various cabins/rooms/spaces in the accommodation block of offshore vessels, the technical solutions (module candidates) can be thought of comprised of these cabins only. In other words, the preliminary technical solutions are crew single module, crew double module, officer's module, captain class module, office module, store module and so on. Although different types of office spaces and stores can have diverse sizes and dimensions; for a modular product platform, it will be beneficial to have a standard office module (or a store module in case of stores) which is scalable to comply with customer requirements. This assumption is reflected in the following step where all the office spaces and storage spaces are considered as a single technical solution.

5.2.3. Generate module concept

Module indication matrix (MIM) is used here to assess each technical solution against the modularity drivers. The modularity drivers are already shown in chapter 1. Here they will be used to evaluate which technical solutions are the most suitable ones for further investigation. Like the QFD in step 1, module candidates will be scored on a scale of 1 to 9 where 9 denotes strong module driver, 3 being medium modular driver and 1 being weak module driver.

Module drivers		Technical Solutions	Crew single	Crew double	Officer class suite	Captain class suite	Various office spaces	Various storage spaces	Dayroom	Galley	Mess	Linen store	Ship laundry	Hospital	Provision stores	Total
Development and design	Carryover	⊙	⊙	⊙	⊙	⊙	•					⊙	•			32
	Technology evolution															0
	Planned design changes	⊙	⊙	•	•											8
Variance	Different specification			⊙	⊙											12
	Styling			⊙	⊙											6
Manufacturing	Common unit	⊙	⊙	⊙	⊙	⊙	⊙					⊙				33
	Process/Org.															0
Quality	Separate testability	⊙	⊙	⊙	⊙											12
Purchase	Supplier avail.															0
After sales	Service/main.	⊙	⊙	⊙	⊙	⊙	⊙					•				43
	Upgrading	⊙	⊙													6
	Recycling	•	•	•	•											4
	Total	3 7	3 7	2 6	3 2	9	6	0	0	0	0	7	1	0	0	

⊙= strong module driver (9 points); ⊙=medium module driver (3 points); •=weak module driver (1 point)

Table 4: Module Indication Matrix (MIM) for module candidates for accommodation block

One thing to be noted here is that Ericsson and Erixon established the module drivers for this procedure with mechanical systems in mind. A machine or an engine has smaller constituent parts for which these module drivers can be easily applied and investigated for possible modular solutions. In this case for offshore accommodation, some of the module drivers such as supplier availability, process/organization and technology evolution do not hold much relevance because of the proprietary nature of this modularization project. Another reason is that the module candidates are an almost homogeneous mixture of cabins that are not so different unlike the different parts in a machine.

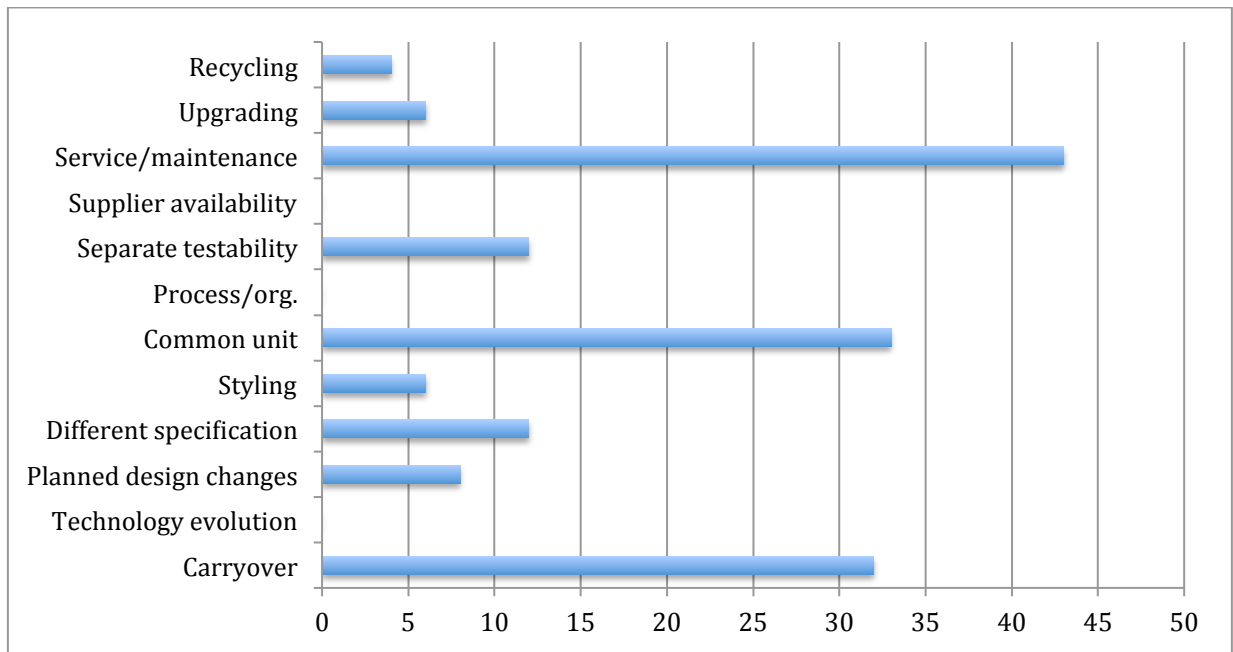


Figure 18: Module driver profile for accommodation block design

From the MIM, the module driver profile for the accommodation block has been derived. The figure shows large totals for service/maintenance, common unit and carryover. This signifies a product that will not experience much technical development or drastic changes in specification; rather emphasizes prolonged service life and a sense of maturity in terms of an industrial product.

Depending on the ship type, the number of parts (cabins) to be assembled for the accommodation block is estimated to be around 20~30. This implies the total different types of cabins, not the total number of cabins in an offshore vessel. According to the MFD™ approach, the ideal number of module can be determined by taking the nearest square root of the total

number of parts. So, in this case, the ideal number can be in a range of 4-5. The five technical solutions having the highest module driver scores are:

- Single crew
- Double crew
- Captain class suite
- Officer class suite
- Office spaces

Next, grouping of technical solutions with module candidates is performed. Grouping is an essential part of the MFDTM approach since it dictates which constituent parts will be put together so that sufficient autonomy is maintained for individual modules. Unlike a mechanical system, these technical solutions are completely separate entities and cannot be grouped to form a larger module. However, it might be possible to cluster similar type of modules to form a grand block of modules; which will in essence decrease design variables and facilitate easier installation of the modules. But MFDTM approach does not specify any guideline for this kind of clustering. Nevertheless clustering of similar modules can be a way of thinking especially for larger offshore vessels where there are too many modules to keep track of.

5.2.4. Evaluate module concept

Once a modular concept has been generated, it is most important to determine the interfaces between the modules since fixed interfaces are a condition for successful parallel activities [Ericsson et al. 1999]. For the accommodation block, interfaces can be easily standardized across the platform since the modules are homogeneous in nature. Some interfaces can be standardized as:

- Geometric interface: This will dictate how the module can be placed on the decks of offshore vessels. Special structures may be required for supporting the modules. Having a standardized geometric interface will make it easier to apply the same module over a range of vessels.
- Energy-transmission interface: The modules have to be powered by central generator-switchboard system to provide electricity to inside facilities. This can be standardized for all modules of different types, as transmission lines are usually the same for a vessel.
- HVAC interface: To connect the module with central HVAC system, an interface is required.

- Water and sewage: For supplying/withdrawing water to/from the wash cabinets of the modules, a water and sewage interface have to be placed.

5.2.5. Optimize modules

Here, module specification is created. After completing the module specification form, detailed design work for each module can begin. This basically implies internal design of the modules. Since the focus of this thesis is the modules themselves, not the internal design/arrangement of them and the fact that some of the modules are already fixed to STX specifications (will be clarified in the next chapter), this step will be left out in this regard.

Nevertheless, from this procedure the outcome is that the following spaces should be strategically modularized:

- Single crew
- Double crew
- Captain class suite
- Officer class suite

The MIM also implies that other cabins/spaces (candidates for modularization) should not be modularized apart from the aforementioned four categories of spaces because of the nearest square root rule. For a fitting production strategy, not all the spaces have to be modularized; rather there should be a balance of modular and non-modular scope of construction. The idea here is to identify those, which are the most appropriate for modular building. Since the weights assigned to different criteria for evaluating the technical solutions are subjective, the method can generate different results according to the design rationale/perspective of managerial body. For the next phase of this thesis, said spaces will be treated as pre-defined modules for subsequent application on different types of vessels.

6. Decision support system

The term “Decision Support” seems rather intuitive and simple; it is in fact very loosely defined. It means different things to different people and in different contexts. Also, its meaning has shifted during the recent history [Bohanec, 2003].

In simple words,“ DS means helping you to make good decisions by understanding the effects of all the alternatives. It allows you to answer the question, ‘What will happen if...?’, for a whole range of scenarios.” [SRI-Online]

Decision support frameworks are developed to aid the decision making process for business or organizational activities. These frameworks serve the management, operations, and planning levels of an organization and help to make decisions, which may be rapidly changing and not easily specified in advance. Usually, support systems are computer based information systems which gathers information from previous projects and/or other inputs and this information is presented in models visualized in a user interface so that the user can adjust the input data to reflect the current needs and develop a suitable solution for the problem. DSS support, rather than replace, managerial judgment. This kind of support system is intended to improve the design process through the use of computers having direct access to databases and reduction of repetitive work on the designer's part. Design can be captured as a decision-making activity where the principal role of the designer is to make decisions based on multilevel information, hard and soft information, multiple measures of merit, non-singular solutions, and "satisficing" but not necessarily optimum choices [McClure, 1990]. According to Mistree, “design involves a series of decisions some of which may be made sequentially and others that must be made concurrently “[Mistree et al. 1990].

Designing ships involves processing a lot of design requirements while considering an overflow of guidelines, regulations, restrictions and so on. The owner has to make a lot of design choices when building a new vessel. How to bring all these factors together and to make informed decisions on them is where the designer comes in. By developing a framework to tackle all these issues, the designer can provide the owner with a logical step-by-step breakdown of every decision in such a way that each effect can be quantified, thus affording the owner the resource of being able to quickly compare alternatives that would have not been readily available to him/her in the past [Katsoufis, 2006].

In essence, a decision support system has three components:

1. Knowledge-base
2. Model
3. User interface

There are different kinds of knowledge bases available for different applications such as rule-, model- or case-based system. Particularly for this discussion, the most suitable knowledge base would be the rule-based one and to some extent the model-based systems. Rule based approach follows the notion of “if condition then consequence”. At each iteration/step, the system examines the entire set of rules and considers only the set of rules that can be executed next [Blecker et al. 2004]. In accommodation block designing, rules can incorporate important knowledge of constraints like compatibilities between different types of spaces (such as crew accommodation and galley), strategic decision (such as hierarchy of space allocation, captain’s cabin should be close to the bridge), classification regulations (such as minimum amount of space for different cabins) etc. These factors can also be associated with model-based approach specially the logic based configuration system. Here, logics are tools for representation and reasoning with knowledge. These logics will bridge the underlying rules and constraints to complex concepts or alternative ideas for solution [Kawser, 2011].

6.1 Module hierarchy

Common modules (CM)	Fixed modules (FM)	Accommodation modules (AM)
Mess	Public WC	Single crew
Conference room	Cleaning locker	Double crew
Ship offices	Storage space	Officer’s cabin
Gymnasium	Staircase	Captain’s cabin
Wardrobe	Engine casing	Other ranked cabins
Stores		
Dayroom		
Galley		
Provision spaces		

Table 5: Module hierarchy

In this research work, the spaces onboard offshore vessels are classified into three classes, namely common, fixed and accommodation modules.

From the MFD™ approach (Chapter 5), it has been seen that crew cabins (single and double) and captain/officer class cabins should be modularized for a fitting modularization strategy for offshore vessels. Here, these cabins are labeled as accommodation modules, which will be treated as fixed dimensioned modules by the system. Next, the fixed modules denote the spaces, which are more or less locked in terms of inside area. For example, areas for public WCs or cleaning lockers do not generally depend on the number of crew or the size of the vessel. Rather they are placed on individual decks as common facilities. From the data gathered in SBS D Ship-4C project (summer internship) [Kawser, 2011], it is apparent that these cabins contain almost the same amount of space in a broad range of vessel. Hence they can be pre-fabricated as complete modules ready to be placed onboard vessels without the need of scaling.

Lastly, the common modules are not modules per se. Since these spaces depend on the number of crew and many other factors such as location (deck, level), mission requirements etc., they cannot be modularized like the accommodation or fixed modules. The fact that these spaces can be too large to modularize them is another argument to keep them non-standardized. The output of MFD™ method also suggests that these spaces should be left as-is. Although, data from SBS D project suggests some possible trend for designing these spaces, they are better suited for designer intervention. Nevertheless, the following DSS will calculate the required area for these spaces based on crew size and available data from the vessel database established in the SBS D project. It will interpolate/extrapolate relevant data of previously built vessels and scale it as necessary.

6.2 Base structure

The software will itself be of modular nature (software modules can be added as per users preference).

- Class regulations (different modules /libraries for different classification society)
- Vessel database module or Fixed value chart (the system can consult the vessel database if added to the platform or it can obtain values from a fixed chart for space allocation just like fixed modules)
- Visualization module
- Arrangement template module (arrangement of the modules will be done based on some specific templates. Extra templates can be added for different types of vessels)

STX specification for crew modules:

- Single crew module: 9 m² (ballpark figure)
- Double crew module: 13 m² (ballpark figure)

Officers cabin specification:

Officers' cabin spaces are fixed based on their ranks.

- Module size for Captain's cabin (rank 1): Single crew x 3
- Module size for Officers cabin (rank 2): Single crew x 2
- Module size for other ranked cabin (rank 3): Double crew x 1

It means that these modules will be manufactured as standardized double/triple sized single/double crew module. Here, the size of the module will be kept open as long as the user opts for any integer derivative of a single/double crew module for the officer class modules (more on this in Chapter 7). The argument behind this kind of arrangement is that it will be easier to handle sizing and scaling of the larger modules if they are sourced from the same baseline dimensions (in this case the single/double crew module).

6.3 Required input

High-level user input:

- Type of vessel (AHTS, OSCV, PSV, Seismic etc.)
- Selection of classification rules (DNV, BV, LR, GL etc.)

Vessel specification:

- Total available area for accommodation block
- Number of deck
- Maximum allowable size of the modules

Accommodation specification:

- Number of crew
- Number of different ranks other than crew
- Number of personnel in each rank

Arrangement options:

- Selection of common modules
- Number and type of fixed modules
- Special arrangement requirements by the owner
- Selection of arrangement template

6.4 Templates

Arrangement of modules will be done according to standard templates. These templates will be implemented in the system as library. There will be different types of templates depending on the deck level. For example, higher deck levels will have templates specifically designed for accommodating rank officers. On the other hand, lower deck levels will have templates designed mostly for crew accommodation. The designer is free to pick different templates for different deck levels, or he/she can also use the same template for all deck levels. For designing PSVs, this can come in handy because of lack of space and the standardized setup for almost all the decks.

The three major types of modules will have their designated location in each of these templates. Some of the templates may not contain modules from three major categories. For instance, a template for officer's deck may not contain any common module at all. The templates will be based on standard grid system to facilitate the application on different sizes of decks. These can also be updated or modified by the designer if the need arises. Since this will be accomplished in a library format, more templates can be added later by the designer's choice.

Some example templates can be:

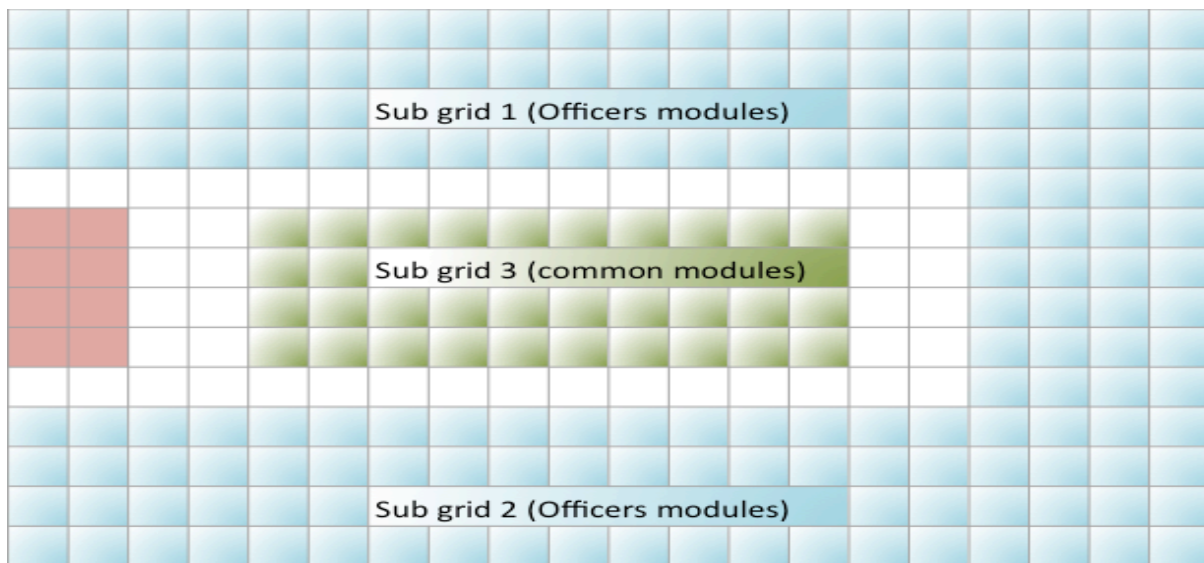


Figure 19: A possible arrangement template for an officer's deck

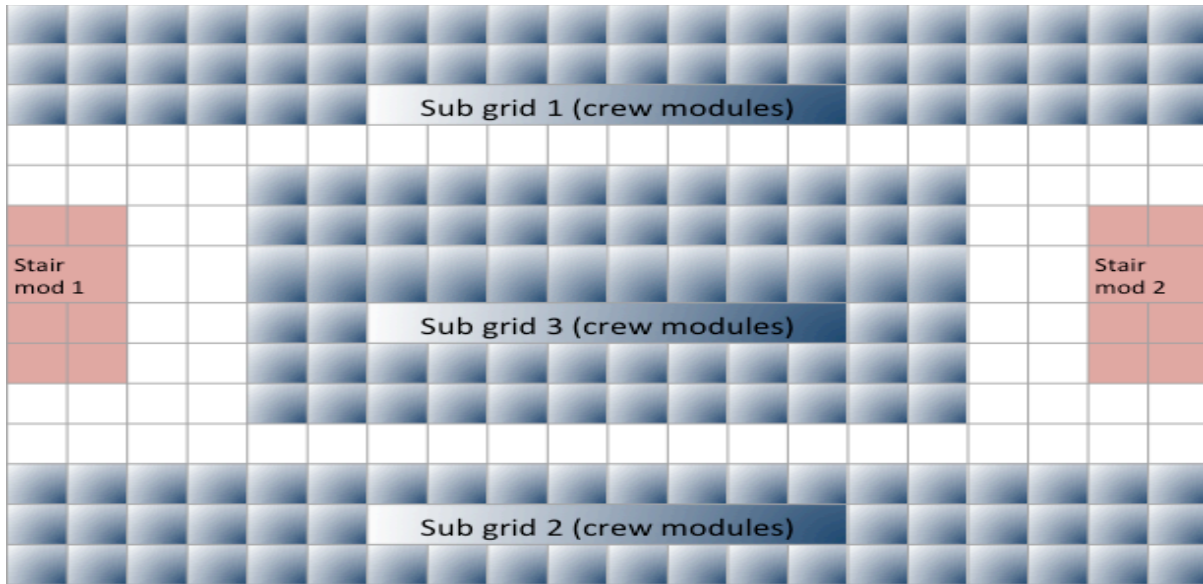


Figure 20: A possible arrangement template for a crew deck

6.5 Major assumptions

- Total available area for accommodation block is known which will be used as an input for this DSF. This number can be derived from data of previous projects or similar type of vessel. It can also be an expert opinion of the designer of the vessel or an educated guess that can be modified later on the DSF if necessary.
- The crew modules are pre-defined and pre-fabricated. They can be scaled in one direction only (STX specification). Apart from the crew modules, every other module can be freely scalable.
- Fixed modules are totally fixed in dimensions and area. Although these modules can be treated as common modules, the initial setup of the system dictates that the dimensions of these modules will be locked to specific values.
- Decks are rectangular and so are the modules. Since, the system will use rectangular grids for calculation and allocation of area for modules, the decks and the modules have to be rectangular in design to complement this system.
- Space allocation is based on minimum space requirement criteria, which can be different for different classification societies or in special cases dependent on owner's specifications.
- Templates are available for overall arrangement of modules on decks. These templates can be modified/updated by the designer. Additional templates can be incorporated into the system if necessary for example when designing a novel vessel.

6.6 Summarized flow

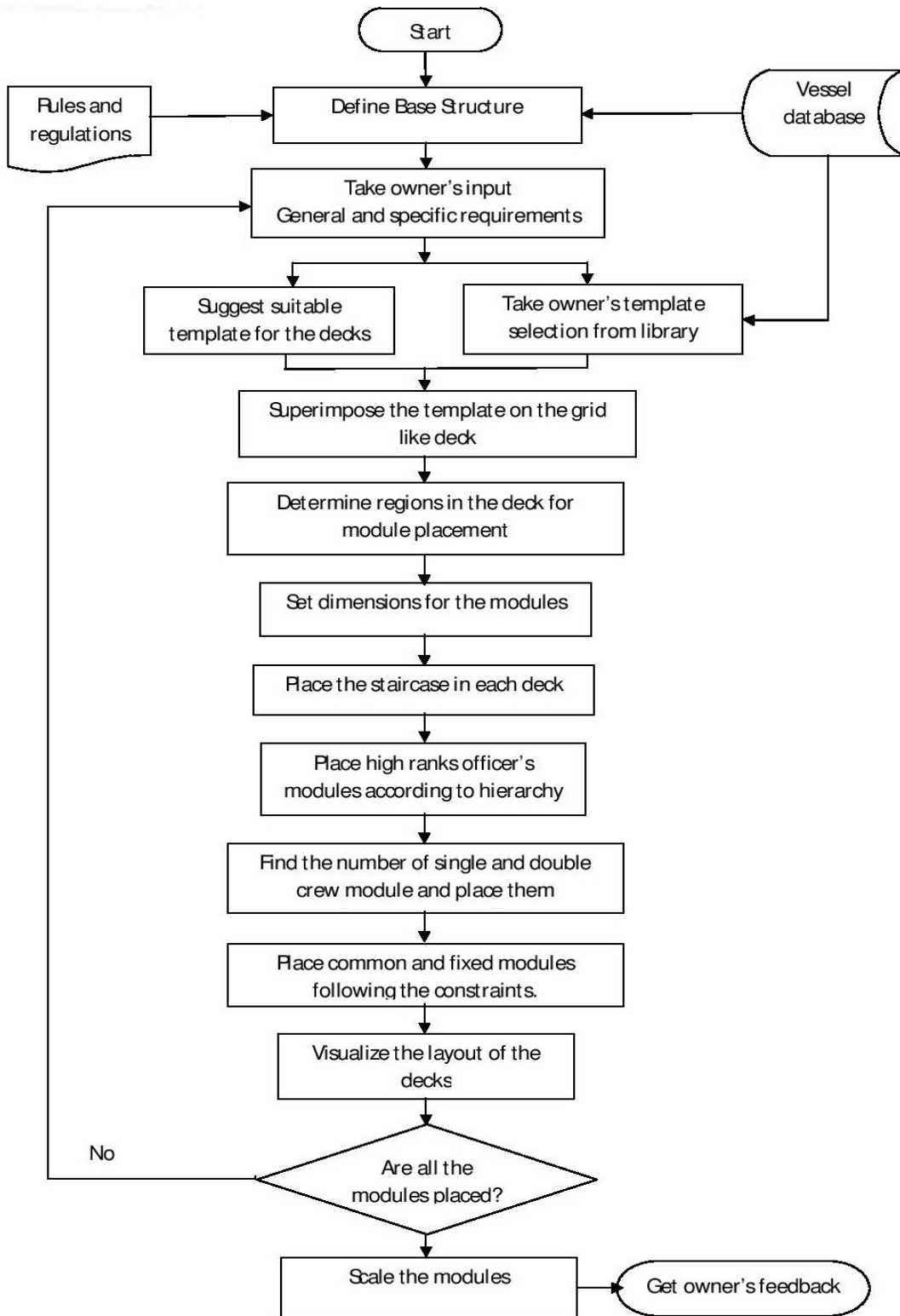


Figure 21: The DSS

6.7 DSS: The steps

The notations used in the system are summarized in the following table:

Number of decks	NDK
Number of different rank	NR
Number of personnel in a rank	NP
Number of crew	NC
Length of a deck in meter	L
Width of a deck in meter	W
Length of a deck in the grid	M
Width of a deck in the grid	N
Deck [M][N]	D
Cumulative Sum [M][N]	C
Single crew module	SC
Double crew module	DC
Officer's module	OM
Common module	CM
Fixed module	FM
Staircase	ST
Current deck	Cd
Number of modules placed in a deck	Mp
Number of unplaced modules	Mu
Number of single crew module	NS
Number of double crew module	ND

Table 6: Notations used in the decision support system

For the sake of simplicity and consistency, these notations will be used replacing the textual expression of variables throughout the decision support framework as well as the examples and case studies in the following sections of this thesis.

Different steps in the DSS will be elaborated in the next pages with necessary mathematical formulations. Major actions in each of these steps will be summarized in separate flow charts for specific steps. The whole system will then be exemplified with an illustrative example at the end of the chapter.

6.7.1 User input phase

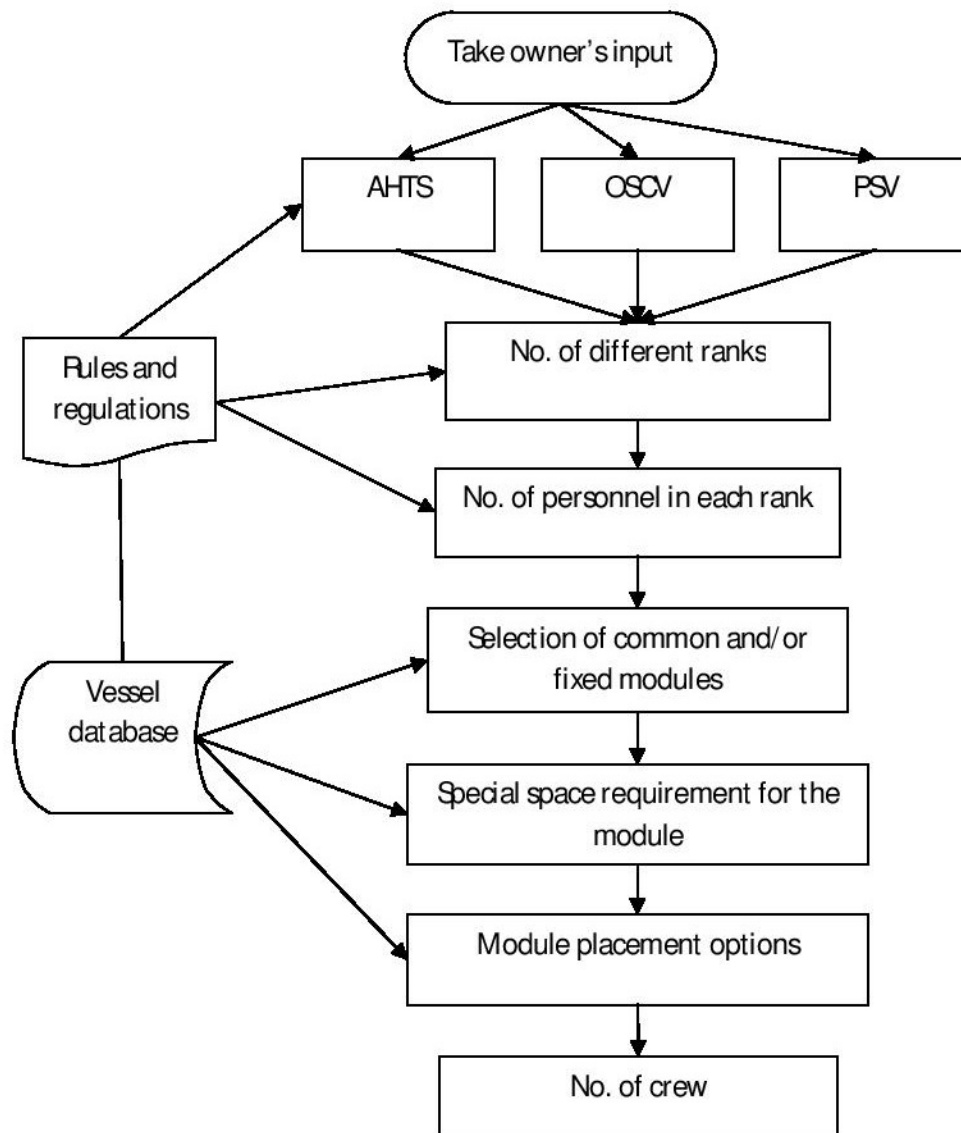


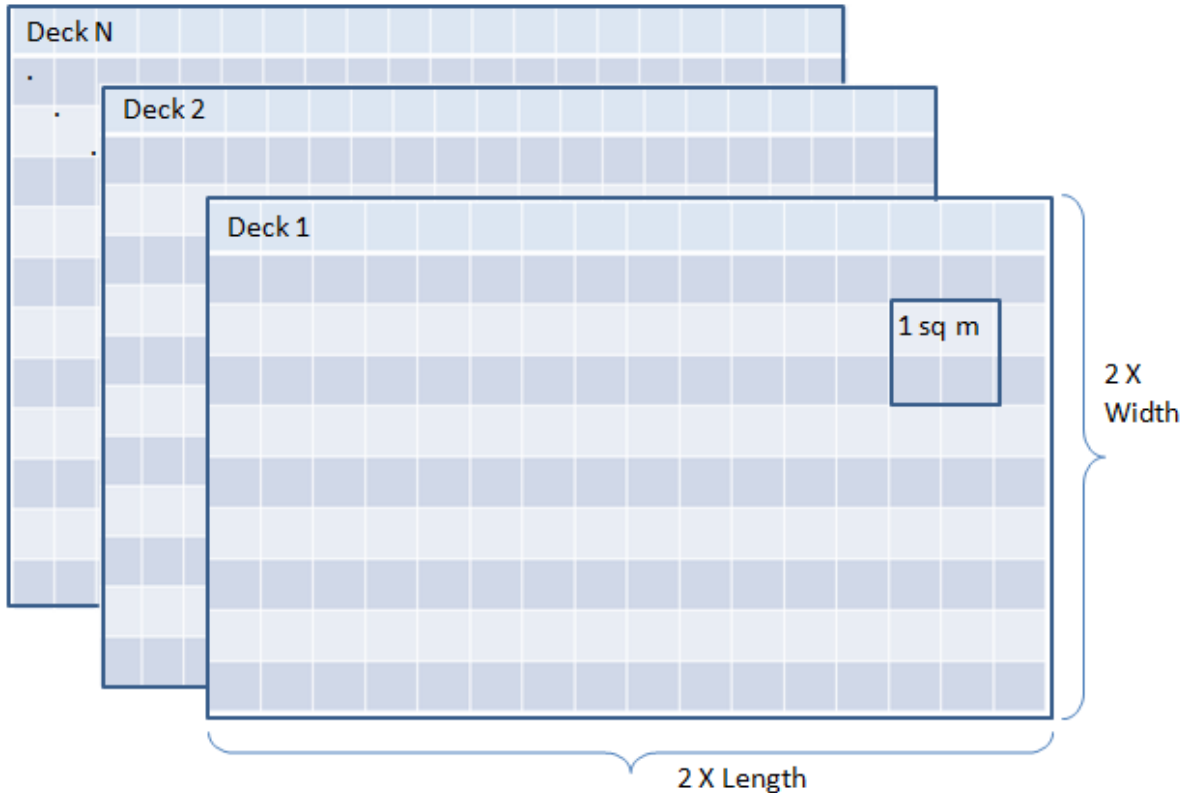
Figure 22: User input phase

6.7.2 DSS initialization: Grid system for the decks

Assume every deck is a $M \times N$ grid, D where area of each cell = 0.25 m^2 . This is used for placing the modules according to the template, user specification, and class regulations. The template is logically superimposed on this. The areas that are already allocated can easily be identified in this grid. Cell area can also be affixed to other values such as 0.5 m^2 or 1 m^2 . For this DSS, 0.25 m^2 seems to be a good enough limit that should give acceptable results.

The following mathematical expressions can be used here:

- $M = 2 \times L, N = 2 \times W$



- $D[m][n] = \begin{cases} 0, & \text{the cell is available for placing a module} \\ 1, & \text{the cell is occupied by a module} \end{cases}$
- A module can be represented as a $P \times Q$ sub grid in D if $P \leq M, Q \leq N$.
- C is used for checking availability of space before placing a module in a deck. This is another grid similar to the grid D . For each deck, there is a C grid. In this grid the cumulative sum of the values in D are stored. From the C values of upper left corner and lower right corner, the current state of a sub grid can be checked; whether or not the sub grid is available for placing a new module. C is updated each time a module is placed in or removed from a Deck.

$$C[m][n] = \sum_{x=1}^m \sum_{y=1}^n D[x][y]$$

For example, the availability of the sub grid (5, 4), (8, 7) can be checked as following:

0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	1	1	1	0	0	0	0
3	0	1	1	1	0	0	0	0
4	0	1	1	1	0	1	1	0
5	0	1	1	1	0	1	1	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

A 4 x 3 and a 2 x 2 module is placed
in the deck

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	1	2	3	3	3	3	3
3	0	0	2	4	6	6	6	6	6
4	0	0	3	6	9	9	10	11	11
5	0	0	4	8	12	12	14	16	16
6	0	0	4	8	12	12	14	16	16
7	0	0	4	8	12	12	14	16	16
8	0	0	4	8	12	12	14	16	16

Cumulative Sum for the deck

$$\text{Let } P = C[8][7] - C[4][7] - C[8][3] + C[4][3] = 16 - 11 - 8 + 6 = 3$$

So the sub grid is not available, it has 3 cells occupied. If P were 0, it would be available.

For any sub grid where (p, q) is upper left corner and (x, y) is lower right corner, $x \geq p$, $y \geq q$:

$$P = C[x][y] - C[p-1][y] - C[x][q-1] + C[p-1][q-1]$$

$$= \begin{cases} 0, & \text{the sub grid is available for placing a module of} \\ & \text{length}(x - p + 1) \text{ and width } (y - q + 1) \\ \text{otherwise,} & \text{the sub grid is occupied by a module} \end{cases}$$

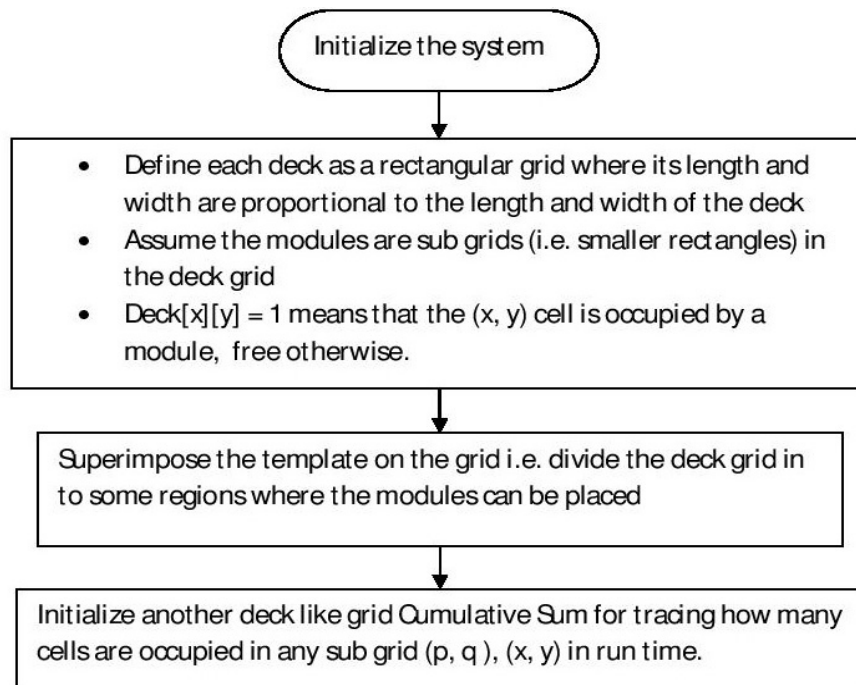


Figure 23: DSS initialization

6.7.3 Template selection

The designer can pick any suitable template from the template library for each deck. The system can also suggest templates for decks based on user input.

For system supported template selection machine learning can be used. For this purpose, some optimality criteria should be set like optimal space usage, symmetry etc. The vessel database can be used to specify these criteria for existing ships. An artificial neural network can be designed and trained with the ships' design. For each template there is a scoring function by which the best template can be determined. When the designer gives some input to the system, it will try to find the best matching template from the database. It will generate more accurate results in real time.

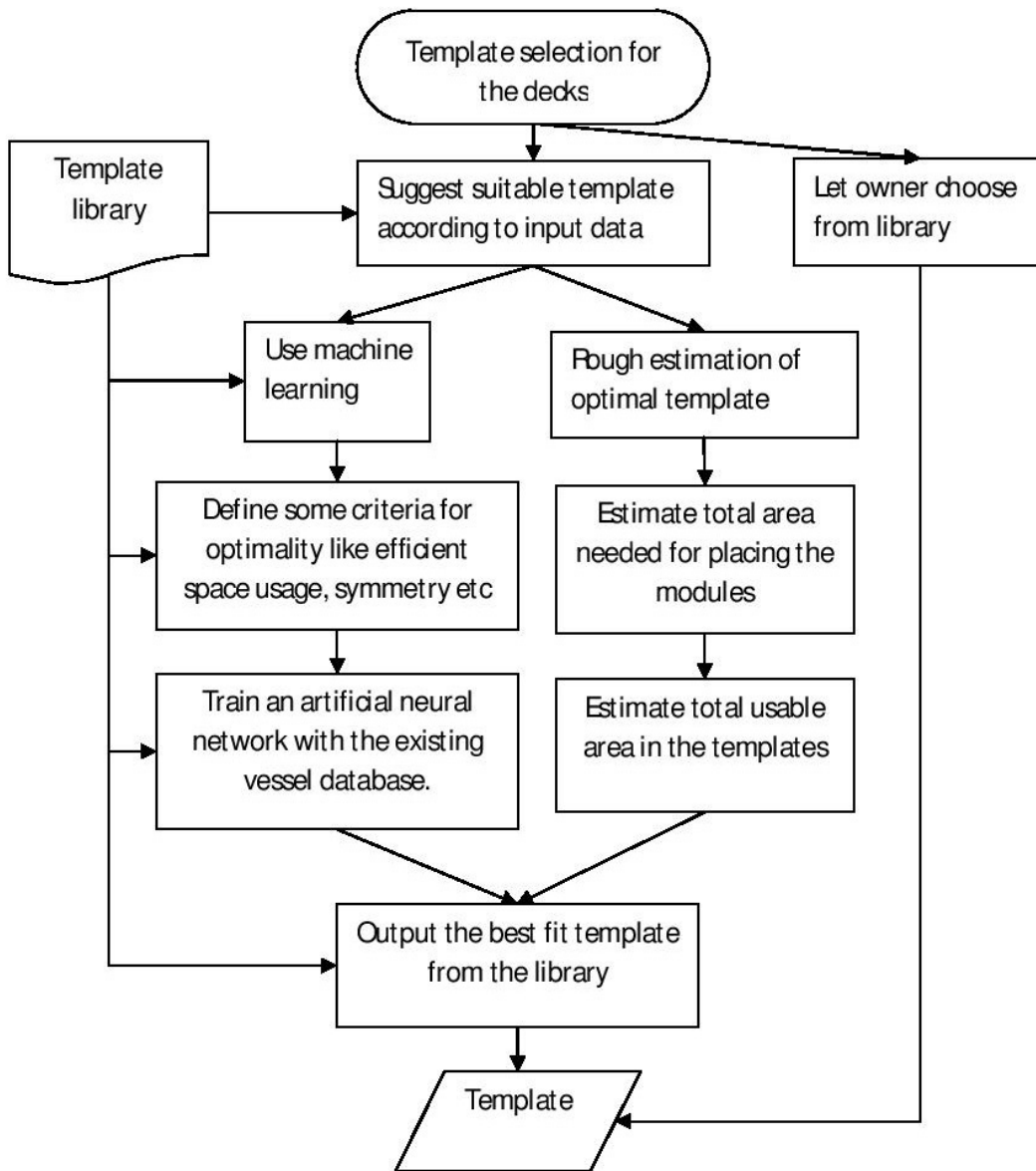


Figure 24: Template selection phase

Another simple and straightforward solution can also be used for the selection. The system calculates the area needed for all the modules. Then it estimates the usable area in each template in the library for the given deck size. One easy and approximate way is to divide the required area for each deck. Then the system will suggest the template whose area is slightly greater than the needed area for each deck.

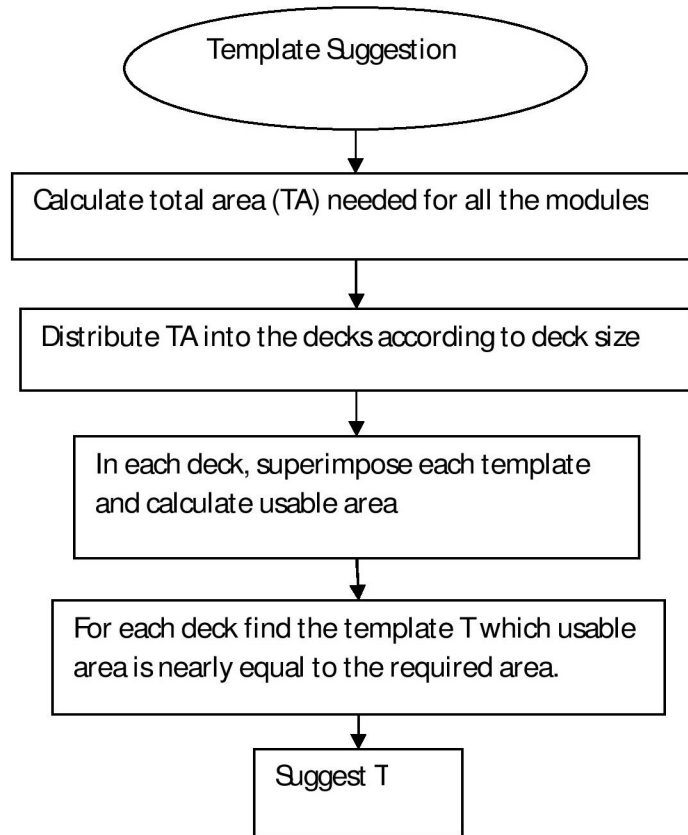


Figure 25: Template suggestion

Different decks can have different types of templates. Total area of D is divided into some smaller grids according to this template. A template is basically a list of rectangles (sub grids) and associated module names.

In this step possible sub grids for each module is determined according to the template, database and other rules (Machine learning can be used for learning from database).

Sub grid	Index of upper left corner	Index of lower right corner	Modules
1	(p_1, q_1)	(x_1, y_1)	SC, DC
2	(p_2, q_2)	(x_2, y_2)	M_1, M_2
3	(p_3, q_3)	(x_3, y_3)	M_3, M_4, M_5
4
5	(p_k, q_k)	$(x_k, y_k):$...

Table 7: Template information table

I.e. single and double crew modules can be placed inside the sub grid between (p_1, q_1) and (x_1, y_1) , modules M_1, M_2 can be placed between (p_2, q_2) and (x_2, y_2) and so on.

There will be additional information like the corridor width, staircase position etc. as well.

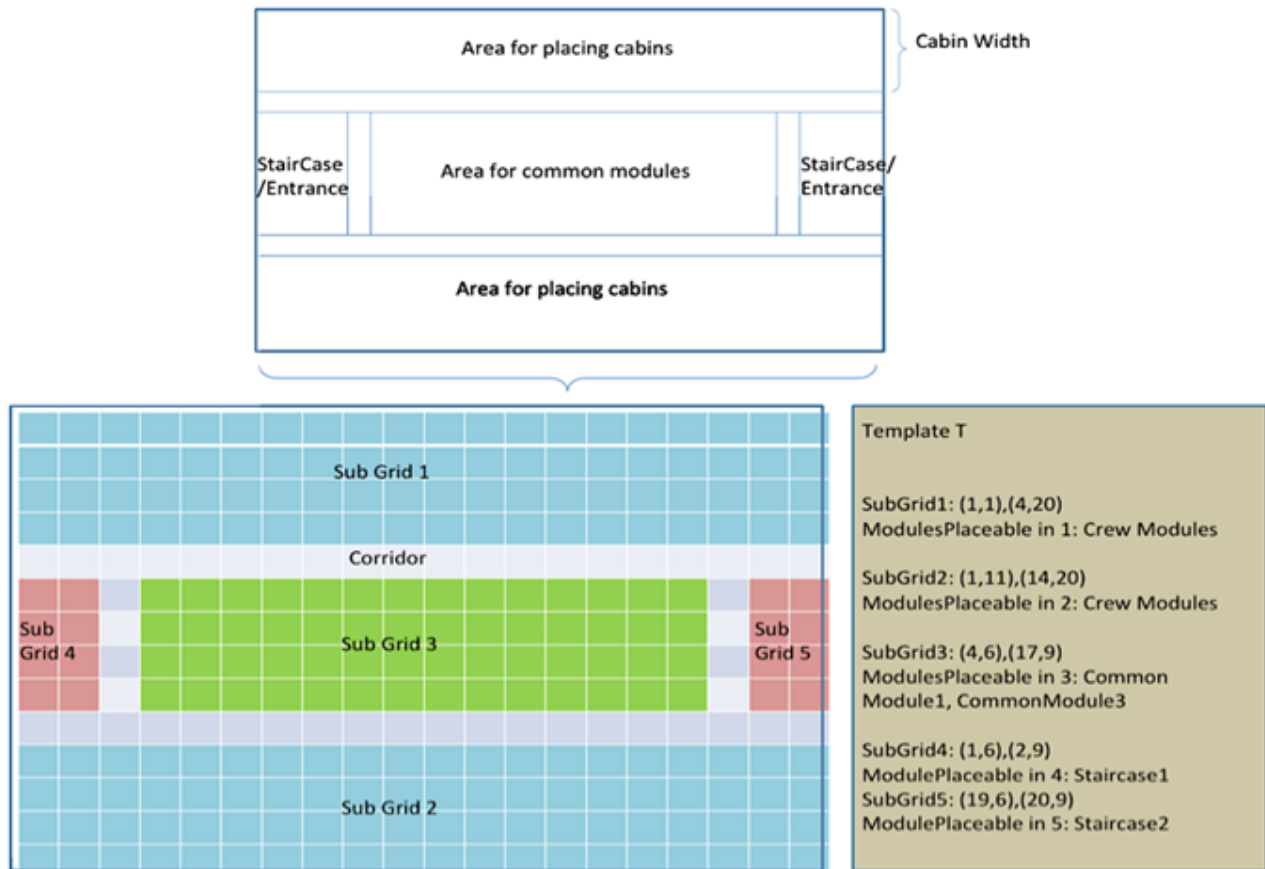


Figure 26: Template

In a template, a specific type of module can appear in more than one sub grid. If there is any special owner's specification or rules dictated by class regulations, then the presence of a module can be restricted to one sub grid only.

6.7.4 Module specification

Attributes have to be defined for each module in terms of grid. In this step the placement of the modules will be determined. Here, the limiting values for that module (dimensions) will also be measured.

- Calculate area/dimension of the common and fixed modules from the database (using machine learning, interpolation etc.) as per user input and rules.
- Calculate length and width of the module

- Set maximum possible length and width
- Set minimum possible length and width
- Generate the list of possible dimensions for the module. Depending on overall size, a module can be placed in different manner. For example: a 30 m² module can be placed as 5m x 6m that is as a 10 x 12 sub grid. Or it can also be placed as 3m x 10m, which will result in a 6 x 20 sub grid. Again, it is also possible to make it like 2m x 15m i.e. 4 x 30 sub grid.

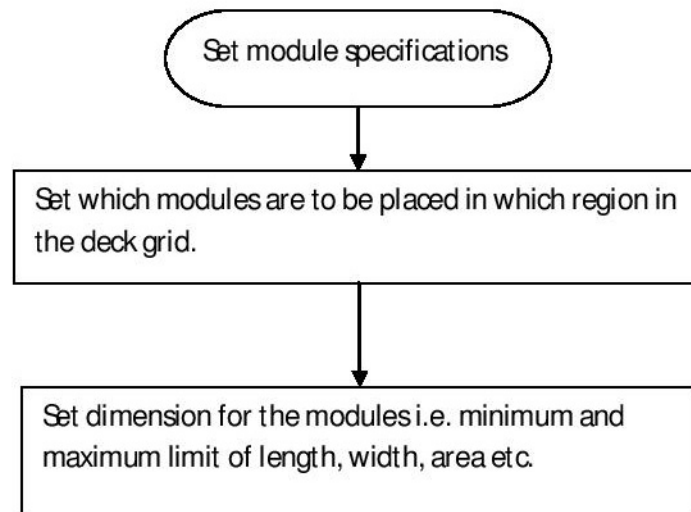


Figure 27: Module specification

Next, the staircases are placed on each deck.

- Mark the area as occupied in D for the decks.
- Update CS for all decks.

6.7.5 Higher ranked officer's module placement

The cabin modules will be placed in this phase. Since there are different types of modules depending on ranks; the system will follow a rank-based order to allocate spaces for these modules.

From the top most deck, it will start placing the modules according to the space hierarchy.

- Set $C_d = 1$, i.e. start from the top most deck.
- For each rank R , $1 \leq R \leq NR$ do
 - i. Set $M_p = 0$. At the beginning no cabin module is placed in the deck.

- ii. Set $M_u = NP$ because for each officer in rank R, a module is needed.
- iii. Select the sub grid S defined in the template for placing officers' modules. There can be multiple sub grids in a deck for placing the modules. We divide the sub grid length to find how many modules can be placed in that sub grid. For simplicity we assume that the width of the sub grid is equal to the width of the module.

$$k = \text{minimum} \left\{ \left\lfloor \frac{\text{Length}(S)}{\text{Length}(OM_R)} \right\rfloor, M_u \right\}$$

- It is possible to place k modules in sub grid S. Then place them consecutively and update D and C for the corresponding deck.
- After placing k modules find the number of remaining modules and/or amount of available space in sub grid S.
- $\text{Length}(S) = \text{Length}(S) - k \times \text{Length}(OM_R)$
- Update the number of placed and unplaced modules.

$$M_p = M_p + k$$

$$M_u = M_u - k$$

- If there are some modules yet to be placed, i.e. $M_u > 0$ then select another sub grid repeat the above steps.
- If there is no more space in the current deck then select the next deck and repeat the process.

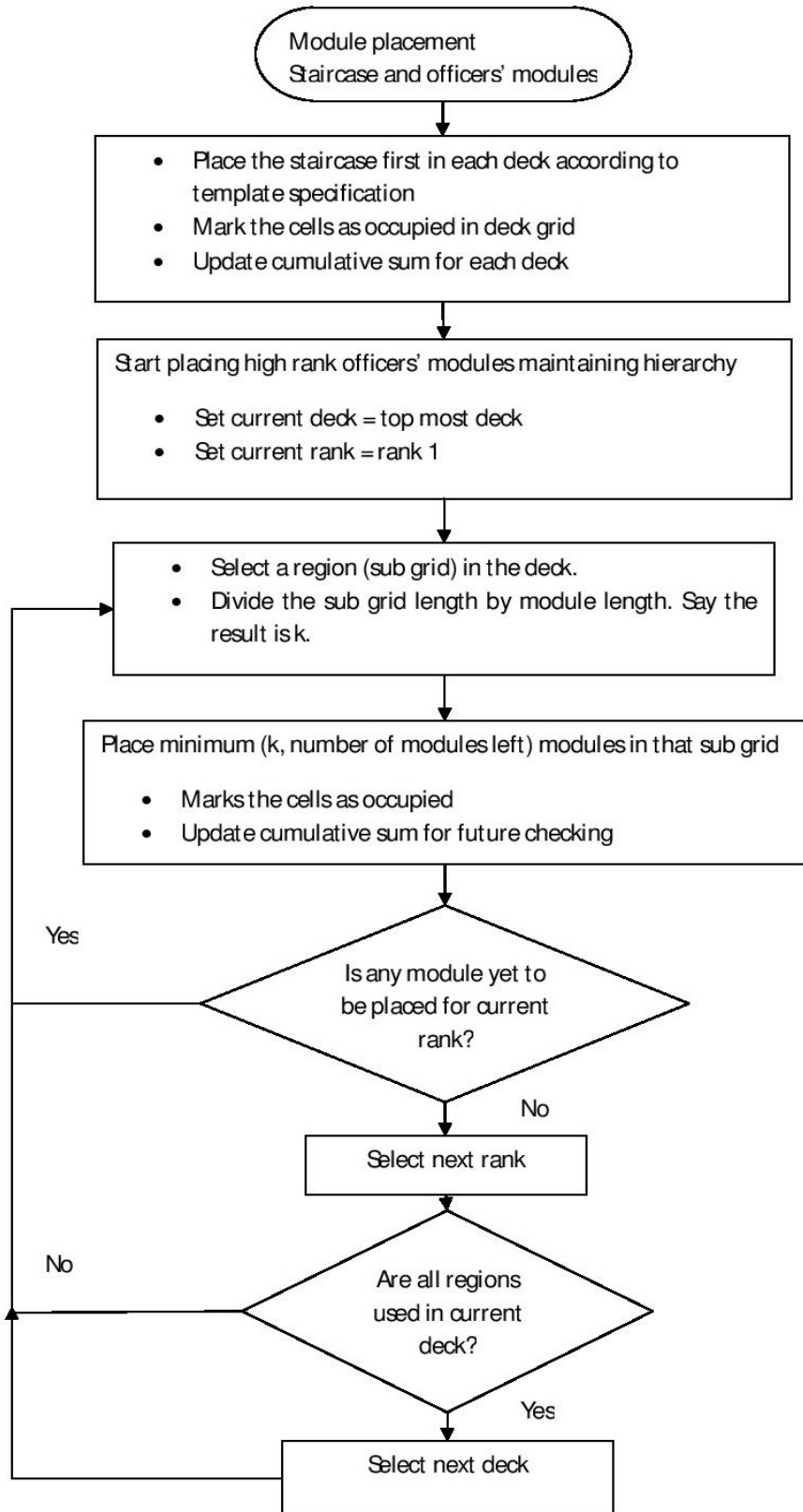


Figure 28: Flow chart for placing officer class modules

6.7.6 Determination of the number of single/double crew modules

Here, the system will calculate the number of single and double crew modules necessary to accommodate the specified crew.

- For simplicity here we also assume that the width of the sub grid is equal to the width of crew module. So the system adds the lengths of the sub grids available for placing crew modules in the decks. If there are r sub grids available:

$$L = \sum_{x=1}^r \text{Length}(S_x)$$

- So roughly $\lfloor L/\text{Length}(\text{SC}) \rfloor$ single crew module or $\lfloor L/\text{Length}(\text{DC}) \rfloor$ double crew modules can be placed in the sub grids. But it may not be possible to place a module if the space is divided into different sub grids. So for safety, some area is subtracted from the total. For r sub grids,

$$L = L - (r - 1) \times \text{Length}(\text{DC})$$

- The system tries to place as much single crew module as possible. If it is not possible to accommodate single modules for all crews, it tries to place double modules. The number of double modules can be calculated from the following formula:

$$\text{Length}(\text{DC}) \times \text{ND} + \text{Length}(\text{SC}) \times (\text{NC} - 2 \times \text{ND}) = L$$

- If $\text{ND} < 0$ then it is not possible to allocate crew modules for given number of crews.

After determining the required number of single and double crew modules, they will be placed on designated decks from top deck; the system will first place the single modules and then the double modules.

- Update D and C for the corresponding deck.
- *If the crew modules cannot be placed in the sub grids place as much as possible and treat the remaining modules as common/fixed module for the current deck.[optional]*

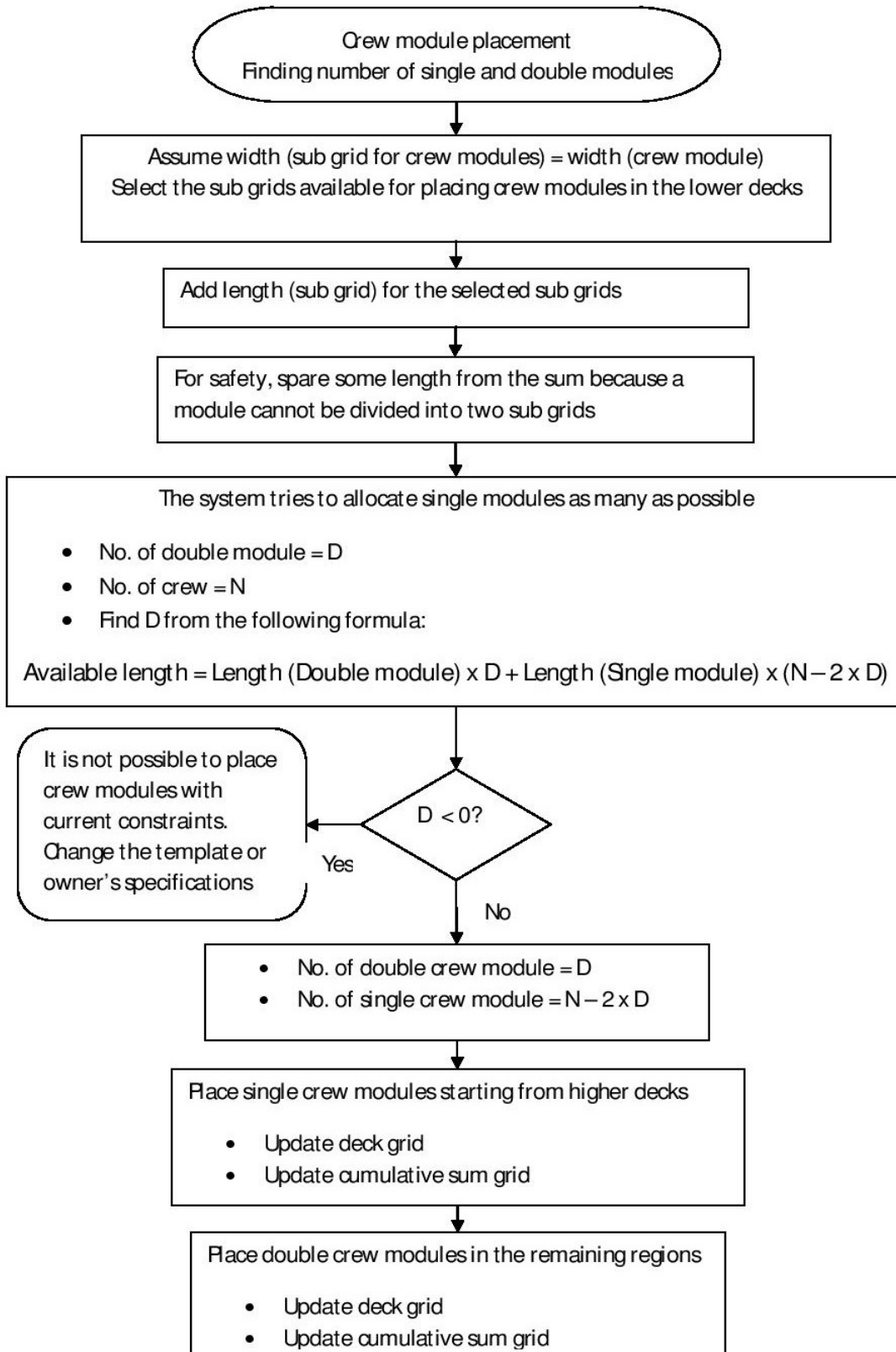


Figure 29: Determination of mix and placement of crew modules

6.7.7 Common and fixed module placement

In this phase a relation between the sub grids and common/fixed modules are established. The relation is defined as a bipartite graph. Finding maximum matching in this graph can be the solution of module placement.

- Make Set A = {All common and fixed modules}
- Make Set B = {All possible sub grids according to previously calculated dimensions for each module in Set A}
- Define a relation $A_x \rightarrow B_y$ if Module A_x can be placed in sub grid B_y
- Find maximum bipartite matching between Set A and B maintaining the following constraints:
 - i. If a module A_x is placed in sub grid B_y , no other module can be placed in B_y
 - ii. At least one side of the module has to be available for corridor.
 - iii. Check if each module is reachable from the staircases. This can be by breadth first search. If all modules are not reachable then try to place the current module to next possible place.
 - iv. Update D and C for the corresponding deck for each placement.

If any module is yet to be placed on board the vessel, but the system do not seem to allocate spaces for those, then the template might have to be altered or additional user input might be required.

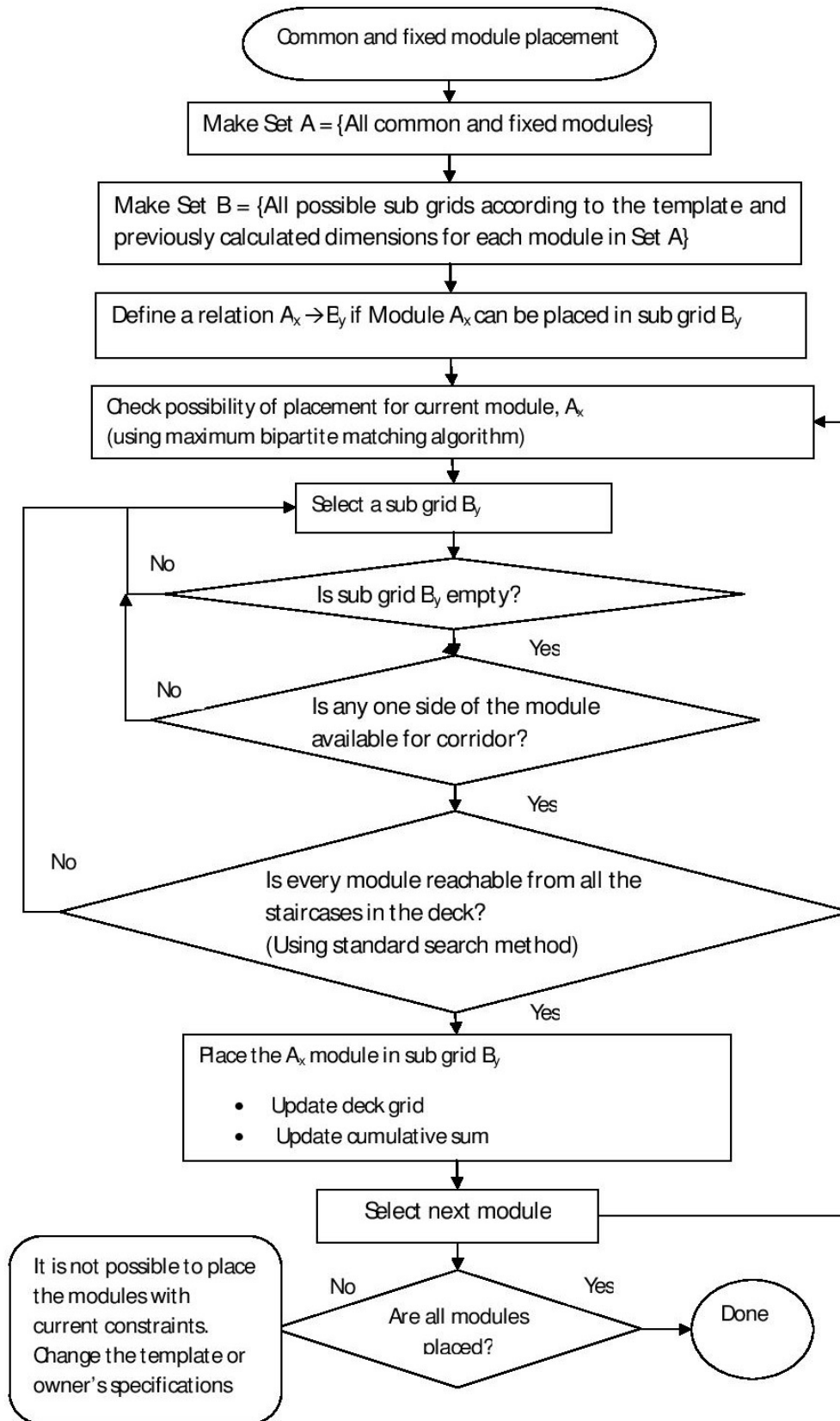


Figure 30: Flow chart for placement of CM and FM

6.7.8 Scaling

- Set maximum possible module size after scaling, $MPS = \text{MaxModuleSize}$
- Set minimum length which can be added, $\text{MinL} = 20 \text{ cm}$ (or according to designer choice)
- Set current rank, $CR = \text{highest rank}$
- Set maximum possible module length after scaling,
 $ML = \text{MaxModuleSize} / \text{Module width of rank CR}$
- Select the regions one by one where at least a rank CR accommodation module is placed.
- For each region, find unused space US (i.e. unoccupied length of that region)
- Divide US by number of modules of same or lower ranks (k) placed in that region:
Extra space per module, $ES = US/k$
- If $ES \geq ML$ then increase the size of each rank CR module:
If current module length + $ES > ML$, set module length = ML
Else increase the module length by ES.
- When finished, find the minimum module size, MS of rank CR from all regions.
Set $MPS = MS - 1$ (assuming different rank modules differ by at least 1sq. m in size)
Set $CR = CR + 1$
Repeat the total process for next rank
- Scaling of fixed and common modules will be done by the designer from a graphical view.
- After manual scaling, a standard BFS method will check whether all the modules are reachable from all the staircases. If it is not, then the designer will be asked to correct the problem or the manual change will be discarded.

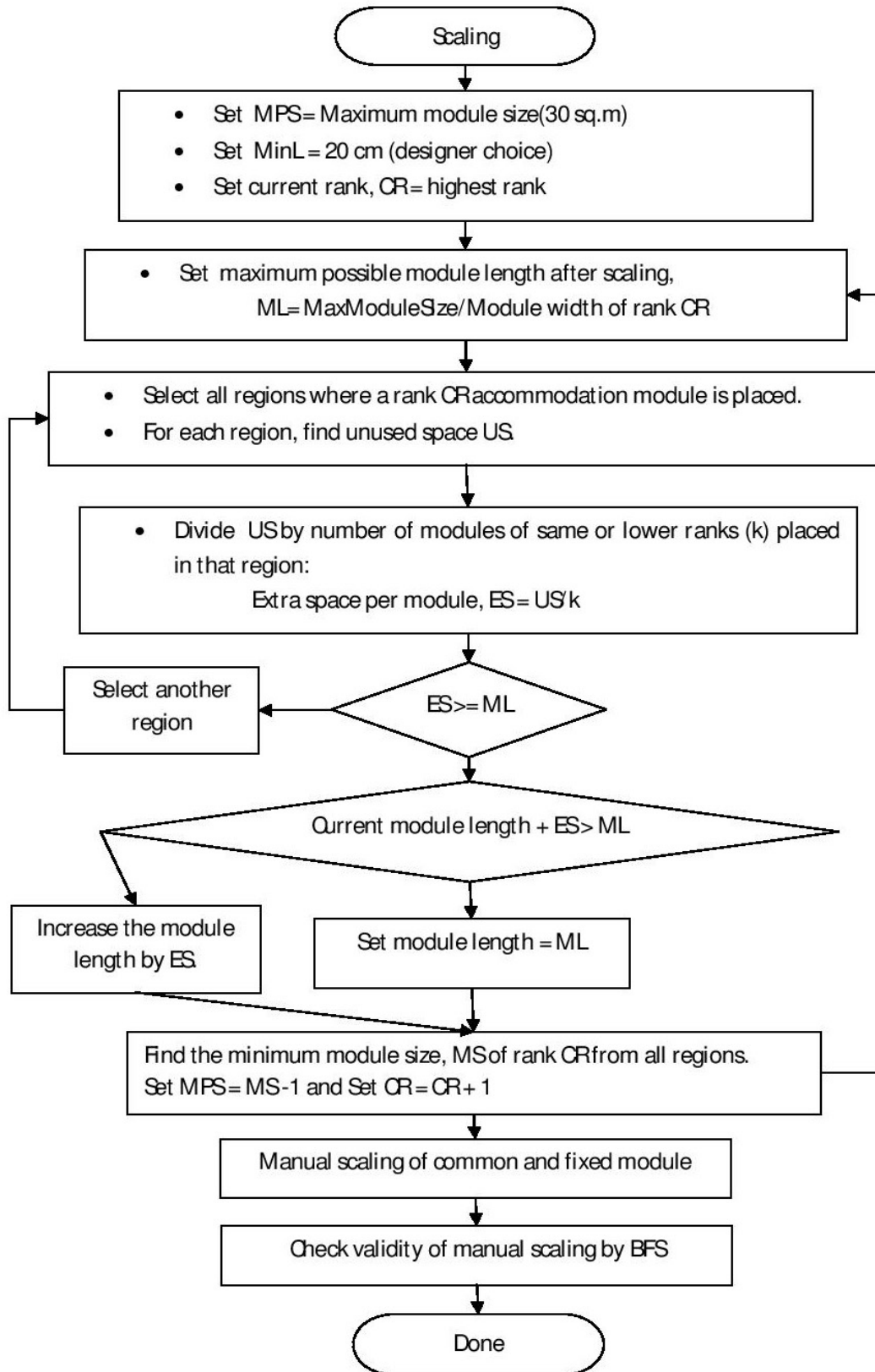


Figure 31: Scaling process

6.8 Relevant theories

6.8.1. Bi partite matching:

A bipartite graph (or bi-graph) is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V ; that is, U and V are independent sets. Bi partite graphs are often used in real cases where it is required to model relations between two different classes of objects.

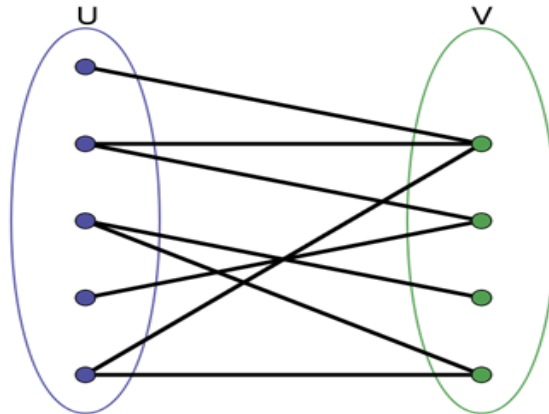


Figure 32: Bi-Partite matching

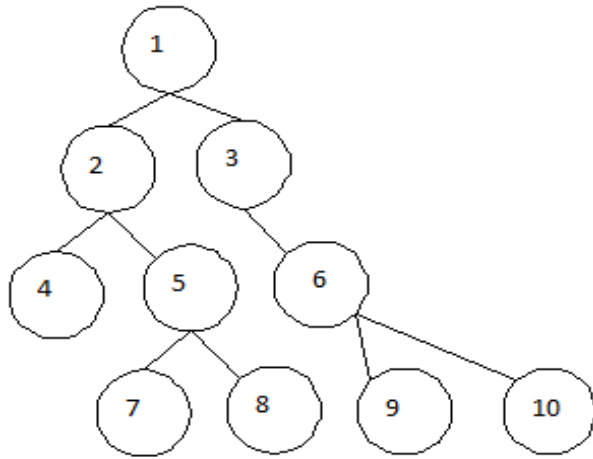
Bipartite graphs are useful for modeling matching problems. An example of bipartite graph is a job-matching problem. Suppose we have a set P of people and a set J of jobs, with not all people suitable for all jobs. We can model this as a bipartite graph (P, J, E) . If a person p_x is suitable for a certain job j_y there is an edge between p_x and j_y in the graph.

Allocating spaces for common and fixed modules in accommodation block design can also be thought of as a bi partite matching problem. Here, not all available sub grids (spaces) are suitable for every common module. For example, certain common modules may have to be placed on upper decks (conference room, owner's office etc.) whereas the others are more suitable for placing on lower deck levels (mess, dayroom etc.). Matching can be done in different ways. A maximum matching is a matching with the largest possible number of edges; it is globally optimal. By reducing the problem to maximum flow in a network, the solution for maximum matching can be achieved [CS787 lecture notes].

6.8.2. Breadth first search (BFS)

BFS is an uninformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. It begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

All child nodes obtained by expanding a node are added to a FIFO (i.e., first in first out) queue. In typical implementations nodes that have not yet been examined for their neighbors are placed in some container like queue or linked list. After visiting the node, it is moved to another container.



The numbers reveal the order by which they are explored.

Figure 33: BFS method

In this system, BFS is used for checking reachability of all modules from the entrance after placing a new module. Here a module is a node. If there is any module, which cannot be explored from all the staircases, the newly placed module should be removed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
6	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1
7	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1
8	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
9	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

Figure 34: Checking reachability by BFS method

6.9 An illustrative example

The DSS will now be exemplified with a simple case study of allotting space for select modules on a deck of a vessel. Here, the goal is to show how the DSS works with user input and what happens in each underlying steps.

User requirements

Number of crew = 20

Deck length = 9 m

Deck width = 6.5 m

Single crew module = 1 X 1.5 m²

Double crew module = 1.5 X 1.5 m²

Corridor width = 0.5 m

Fixed module 1 = 2 X 1.5 m²

Fixed module 2 = 1 X 1 m²

Common module 1 = 1.5 X 2.5 m²

Common module 2 = 2 X 1.5 m²

Staircase 1 = 1 X 1.5 m²

Staircase 2 = 1 X 1.5 m²

Stair case location = center

[These numbers are for illustration purposes only; they may not necessarily represent real life values for offshore vessels]

Modules

Here the dimensions of the modules are set according to the grid. In this case we assume each cell of the grid is equivalent to 0.25 m². So 1.5 meter is equivalent to the length of 3 consecutive cells in the grid.

Module	Name	Possible dimensions in the grid
1	Single crew module	2 X 3
2	Double crew module	3 X 3
3	Fixed module 1	4 X 3
4	Fixed module 2	2 X 2
5	Common module 1	3 X 5
6	Common module 2	4 X 3, 6 X 2
7	Staircase 1	2 X 3
8	Staircase 2	2 X 3

Table 8: Possible module dimensions

Graphical template

The designer can select a template from the library of pre-defined templates. It represents the ratio for dividing the grid into some smaller regions for placing modules according to owner's specifications and rules.

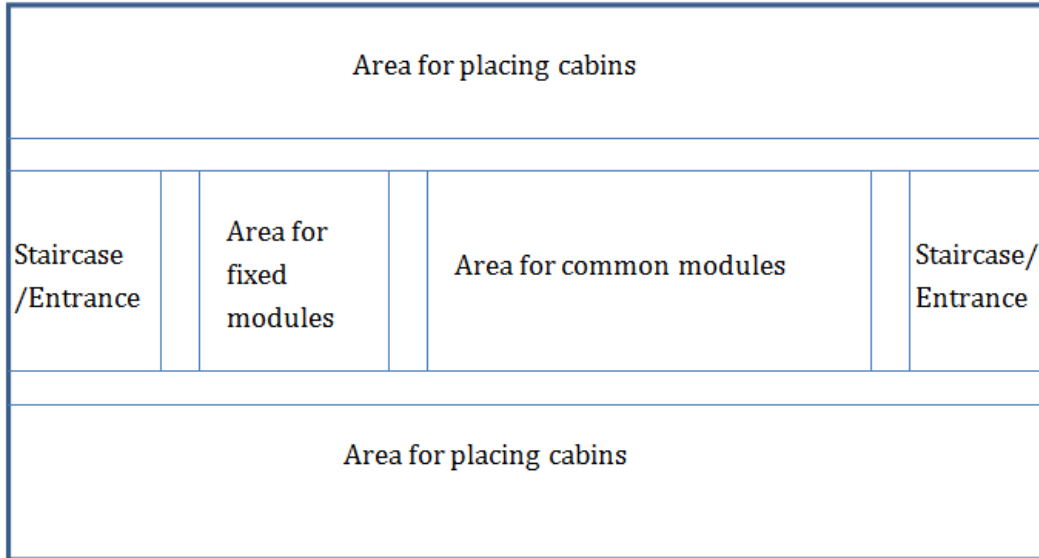


Figure 35: Selected template for module placement

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36: The deck consisting of sub grids

In the above figure the deck is divided into sub grids. Here the blue region is for placing crew modules and its width is equal to the width of a crew module, i.e. 1.5 meter or 3 cells.

The green region is for placing the staircases in the middle. The white region is used for corridors. And the rest of the space in the middle portion is divided into two sub grids by 2:3 ratios for placing fixed and common modules respectively.

Then a list is generated as template T. It has the index of upper left and lower right corners of each sub grid and an associated list of modules that can be placed within the sub grid.

Sub grid	Upper left and lower right corners	Modules
1	(1, 1), (3, 18)	1, 2
2	(6, 1), (8, 2)	7
3	(5, 4), (9, 7)	3, 4
4	(5, 8), (9, 15)	5, 6
5	(6, 17), (8, 18)	8
6	(11, 1), (13, 18)	1, 2

Table 9: Template grid specification

Each deck has its corresponding cumulative sum grid, C. Initially all cells in this grid are set to zero.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37: Cumulative sum grid

At the very first step, the staircases are placed in each deck. This is done to ensure accessibility for each module from the staircases. After placing the staircases, the deck and cumulative sum grid looks like:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
7	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
8	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	4
6	2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	6	8
7	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12
8	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12
9	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12
10	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12
11	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12
12	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12
13	3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	9	12

Figure 38: Grid/Cumulative sum grid after placing the staircases

Finding number of crew modules

Available sub grids for placing crew modules are sub grid 1 ((1, 1), (3, 18)) and sub grid 6 ((11, 1), (13, 18)). Then $L = (18 - 1 + 1) + (18 - 1 + 1) = 36$.

For safety we subtract a length equal to length (DC) = 3. Then $L = 36 - 3 = 33$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
7	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
8	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54
0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54
0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54
0	4	8	11	14	17	20	23	26	29	32	35	38	41	44	47	50	54	58
0	5	10	13	16	19	22	25	28	31	34	37	40	43	46	49	52	57	62
0	6	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	60	66
0	6	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	60	66
0	6	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	60	66
0	7	14	18	22	26	30	34	38	42	46	50	54	58	62	66	69	75	81
0	8	16	21	26	31	36	41	46	51	56	61	66	71	76	81	84	90	96
0	9	18	24	30	36	42	48	54	60	66	72	78	84	90	96	99	105	111

Figure 39: Crew modules are being placed

Applying the formula: Length (DC) x ND + Length (SC) x (NC – 2 x ND) = L

$$\rightarrow 3 \times ND + 2 \times (20 - 2 \times ND) = 33$$

$$\rightarrow ND = 7 \quad \text{So we find number of double modules} = 7$$

Fixed and common modules placement

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
6	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1
7	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1
8	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
9	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

Figure 40: Cumulative sum grid after placing the FMs

In this step, set A: set of modules and set B: set of all sub grids are defined.

A relation $A_x \rightarrow B_y$ is defined if module A_x can be placed in sub grid B_y .

In the following figure a sub set of B is shown. B has all possible sub grids according to the template whose dimensions are equal to the dimensions of the modules.

The matching sub grids are highlighted. Before placing a module, the system checks the feasibility by 3 steps.

1. Is the sub grid empty?

This can be done by checking cumulative sum grid C.

For example, to check if the fixed module 1 can be placed inside (5, 4), (7, 7), it calculates:

$$\begin{aligned}
 P &= C[7][7] - CS[4][7] - CS[7][3] + CS[4][3] \\
 &= 25 - 21 - 13 + 9 \\
 &= 0
 \end{aligned}$$

Now, this sub grid is empty as it has 0 cells occupied.

2. Is any side is empty for corridor?
For this the lower, upper, left and right sides are checked using the above formula in cumulative sum grid.
In this example, 4 sides are available.
3. If this module is placed, are all the modules reachable from the staircases?
To check this, the standard Breadth First Search algorithm is used.

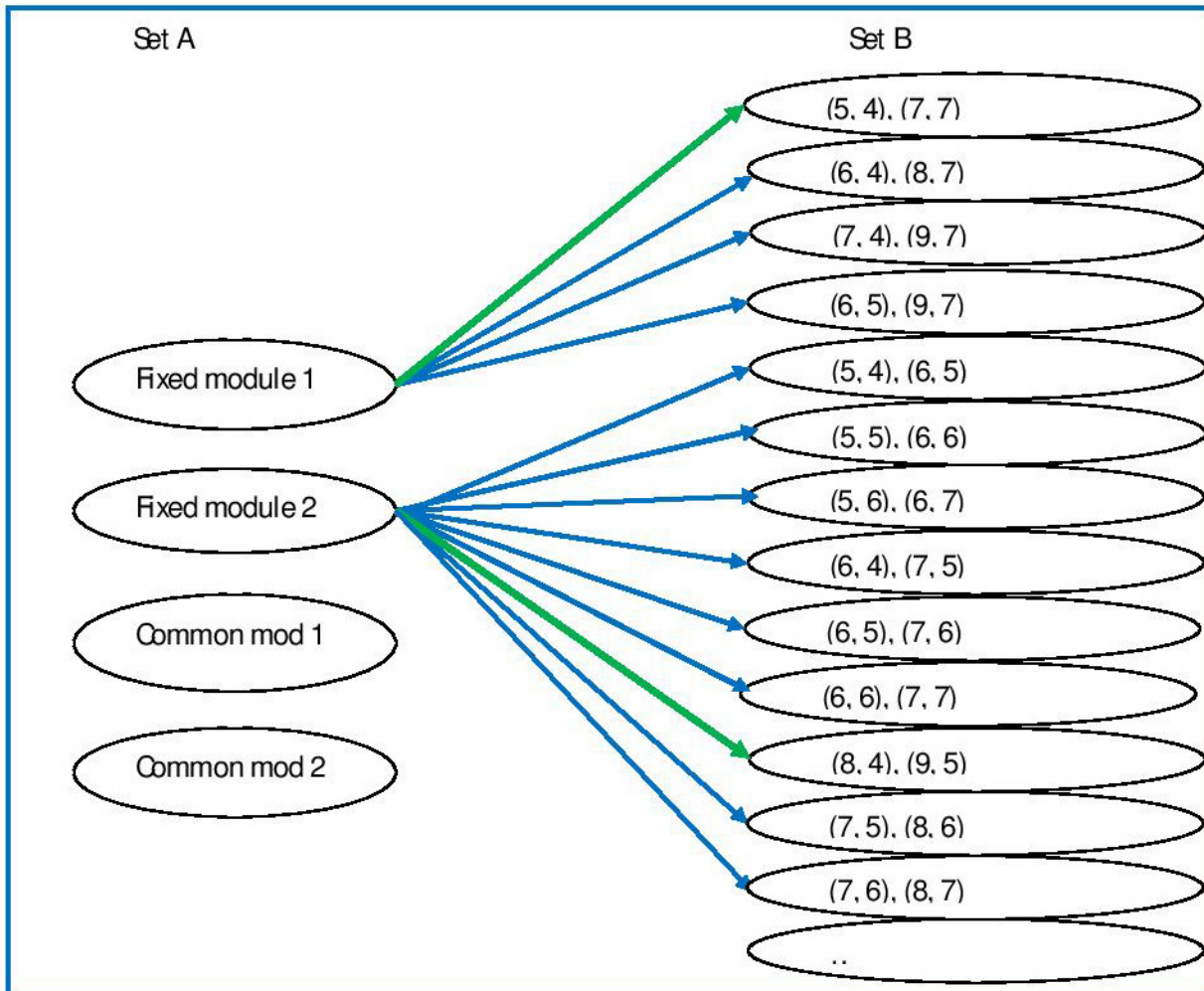


Figure 41: Possible module placement

In similar approach, the fixed module 2 is placed in the sub grid (8, 4), (9, 5). The maximum bipartite matching systematically checks every option of placement for finding the optimal solution.

7. Mock up of a user interface based on the DSS

In this section a mock up of a user interface based on the DSS explained in the previous chapter will be illustrated. Since, the whole system has not been elaborated to a sufficient level of details; a working prototype is not feasible in this stage, hence the mock up will demonstrate how the process can be utilized in real cases. Various input and output phases will be shown following the workflow of the DSS.

The software will have a simple user interface that is uncluttered and easy to use. Step by step windows will appear as the designer proceeds and categorically ask for information regarding the vessel, class, decks, crew, various modules etc.

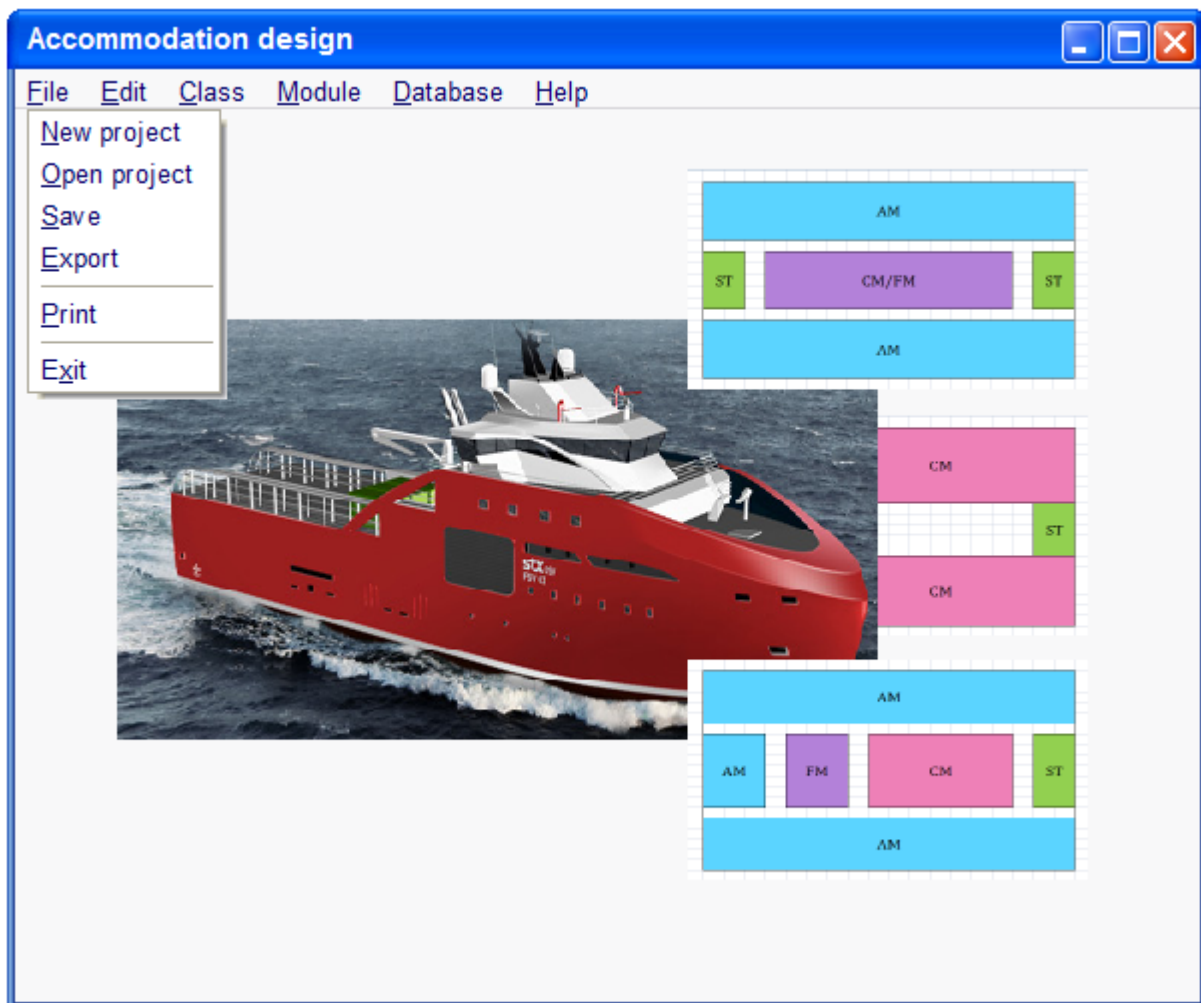


Figure 42: User interface

7.1 Input phase

7.1.1. Selection of vessel type

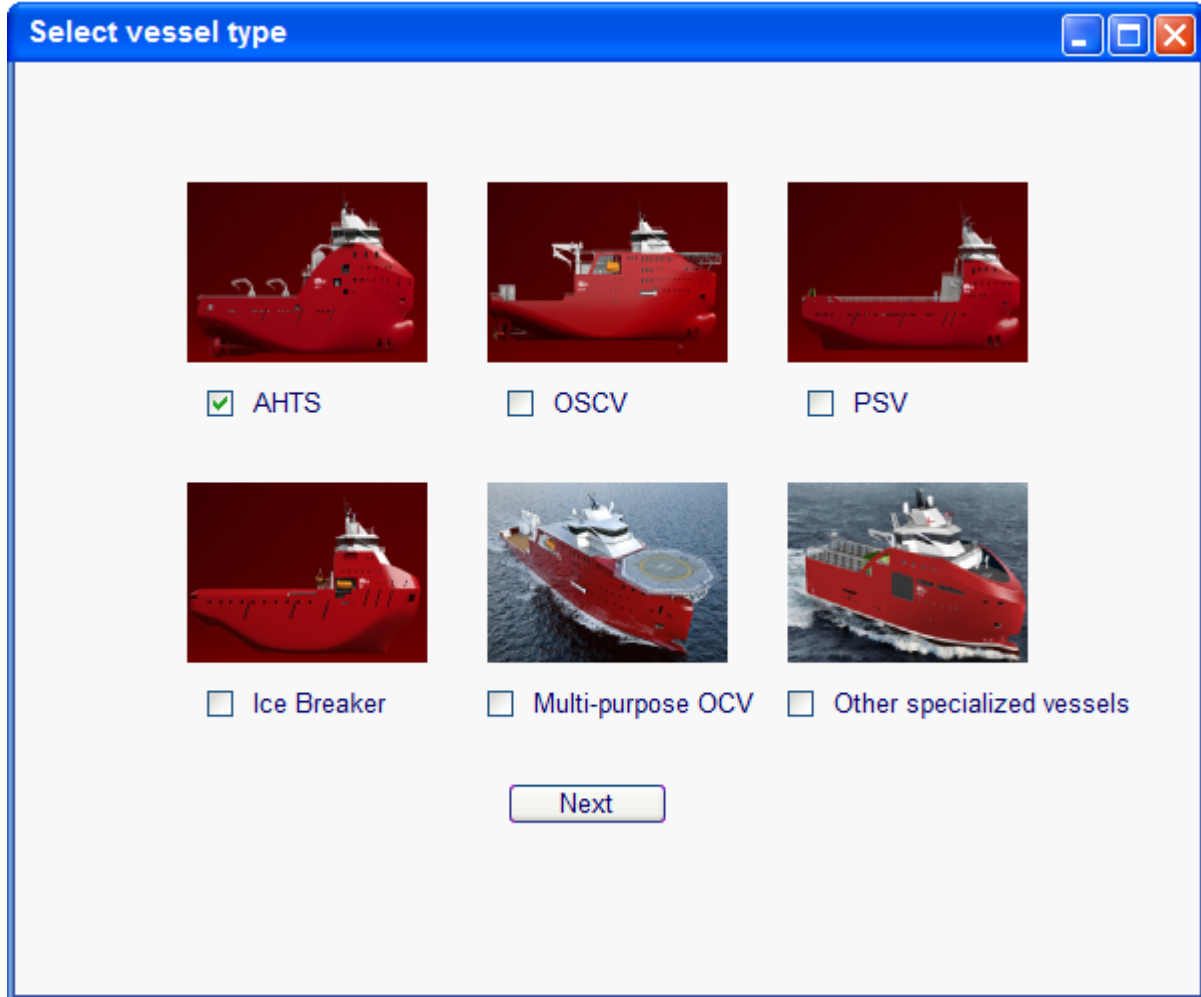


Figure 43: Selection of vessel type

The first step of the DSS is to select the type of vessel. As the research work revolves around offshore support vessels only, the major choices in this area are shown in the window. The vessel database is structured as data-per-vessel-type approach, so depending on the choice; the system will look into the relevant vessel type only. This will speed up the process. When the designer picks one of the options here, the next window will appear with options for class, class notations and database selection.

7.1.2. Selection of class and notations

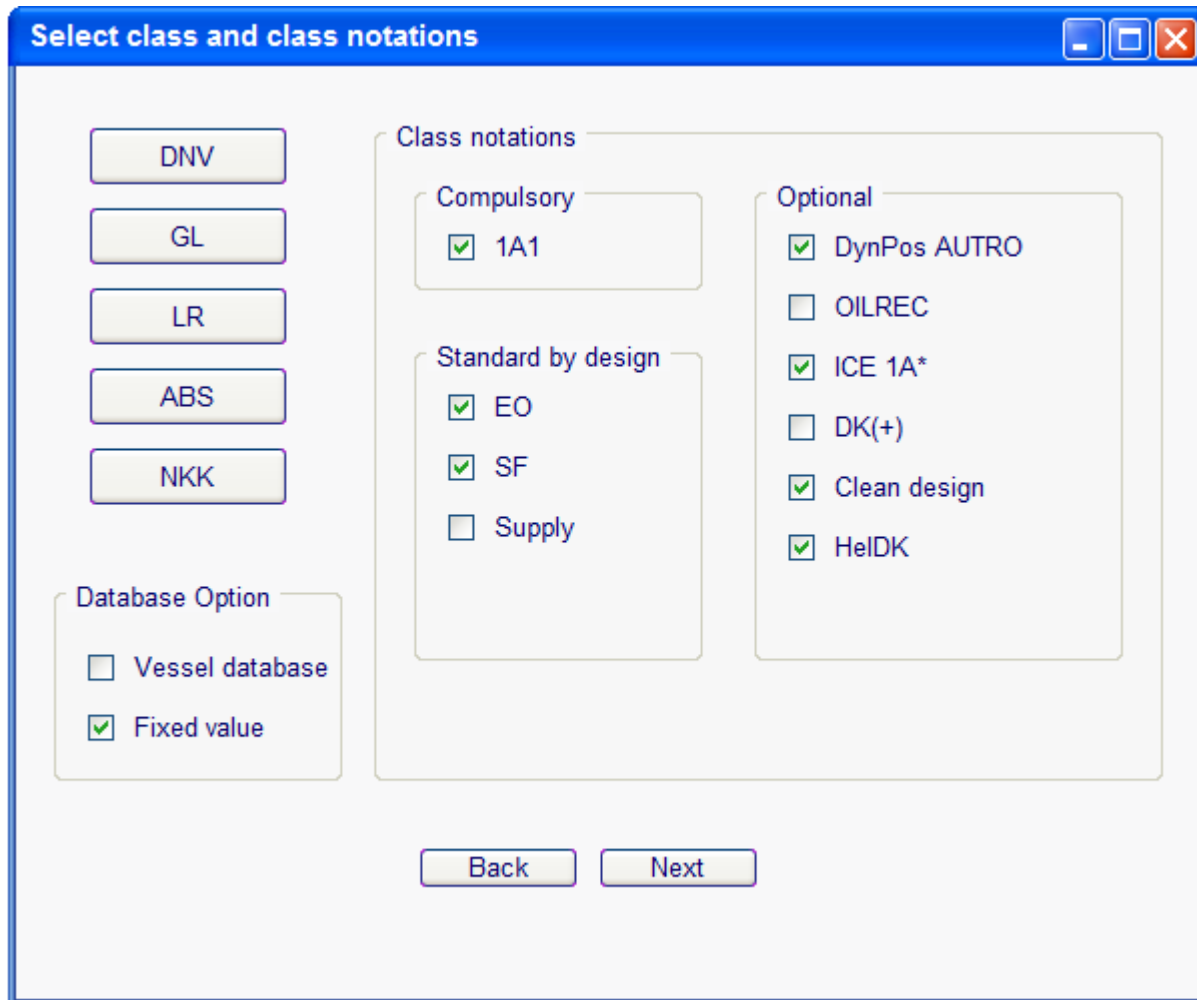


Figure 44: Selection of class and notations

In this window, the user can choose a class from a list of classification society. In this figure, five class modules are depicted namely DNV, GL, LR, ABS and NKK. The entire software is itself a modular one, so it is possible to add class modules containing information from other classification societies later. Based on user input, the class notations section will comprise of the notations specifically from that society. Here, the user can further clarify the notations to be used for this particular vessel. Since, offshore support vessels can have various class notations depending on mission requirements, the rules related to these notations can very much dictate the class requirements for designing the accommodation block.

Another option here in this window is the selection of database type. The user can either choose the vessel database (like the database developed in SBSB program) or s/he can also opt for fixed value approach where he has to provide the necessary information for the system. In this case,

the system won't consult the vessel database, rather use the data provided by the designer. This can be particularly useful for designing novel vessels where an established vessel database is out of question.

7.1.3. Vessel specification

Vessel specification

Number of decks Same area for each deck

Deck specification

	Length(m)	Width(m)	Usable area(sq. m)	No. of staircases
Deck A	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Deck B	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Deck C	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Maximum allowable size of the modules(sq. m)

Single crew module size(sq. m) Length(m)

Double crew module size Length(m)

Template usage

System suggestion

Manual selection

Figure 45: Vessel specification

In this window, the user will provide high-level information on the decks of the vessel and some limiting values for module design. When the field for number of decks is filled with some number (1~15), corresponding deck specification section will show rows for data input for specified number of decks. Here, the base module sizes have to be specified along with maximum module size. Since manual and automated selection of template is possible, an option is given here for the designer to choose between them. Although this can be overridden later in the software.

7.1.4. Customer requirements

Customer Requirements

Number of different officer ranks

Module specification

Number of officer in rank 1 Module size

Number of officer in rank 2 Module size

Number of officer in rank 3 Module size

1 x Single crew module
2 x Single crew module
3 x Single crew module
1 x Double crew module
2 x Double crew module

Crew module specification

Number of crew

Required number of single crew module

Back Next

Figure 46: Customer requirements on crew

Owner requirements for the crew of the vessel will be taken care of in this step. Total number of crew, different ranks of officers and number of officers in each rank will be specified here. The customer will also have option for module sizes for different officer ranks. There are currently five options in the system regarding the size of officers' modules. If the owner wants a specific number of single crew modules for the vessel, then it can be accommodated by filling up the field in this section.

7.1.5. Specification for fixed modules

Number and location of fixed modules will be specified in the following window. These modules are totally fixed in terms of area and dimension, so no options for adjustment will be available here. The designer can use the options 'in each deck' and 'in any deck' for convenience.

Module	Deck A	Deck B	Deck C	In any deck	In each deck	Total
Public WC	<input type="checkbox"/>	<input checked="" type="checkbox"/> 1	<input type="checkbox"/>	<input checked="" type="checkbox"/> 1	<input type="checkbox"/>	2
Cleaning locker	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/> 1	3
Storage space	<input checked="" type="checkbox"/> 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2
Staircase	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/> 2	6
Engine casing	<input type="checkbox"/>	<input checked="" type="checkbox"/> 1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1

Figure 47: Specification for FM

7.1.6. Specification for common modules

Common modules required for the vessel will be selected here from a list. After selecting them, a new window will appear just like the 'fixed module specification' window to place the modules on the decks. Each deck/any deck options are available in this window as well.

Customer requirements for common modules

Selection of common modules

- Mess
- Conference room
- Ship offices
- Gymnasium
- Wardrobe
- Stores
- Dayroom
- Galley
- Provision spaces

Back Next

Common module specifications

Ship offices

<input type="checkbox"/> Deck A	<input type="text" value=""/>	<input checked="" type="checkbox"/> Deck B	<input type="text" value="1"/>	<input type="checkbox"/> Deck C	<input type="text" value=""/>
<input checked="" type="checkbox"/> In any deck	<input type="text" value="1"/>	<input type="checkbox"/> In each deck	<input type="text" value=""/>	Total	<input type="text" value="2"/>

Wardrobe

<input type="checkbox"/> Deck A	<input type="text" value=""/>	<input type="checkbox"/> Deck B	<input type="text" value=""/>	<input type="checkbox"/> Deck C	<input type="text" value=""/>
<input type="checkbox"/> In any deck	<input type="text" value=""/>	<input checked="" type="checkbox"/> In each deck	<input type="text" value="1"/>	Total	<input type="text" value="3"/>

Stores

<input checked="" type="checkbox"/> Deck A	<input type="text" value="2"/>	<input type="checkbox"/> Deck B	<input type="text" value=""/>	<input type="checkbox"/> Deck C	<input type="text" value=""/>
<input type="checkbox"/> In any deck	<input type="text" value=""/>	<input type="checkbox"/> In each deck	<input type="text" value=""/>	Total	<input type="text" value="2"/>

Galley

<input type="checkbox"/> Deck A	<input type="text" value=""/>	<input type="checkbox"/> Deck B	<input type="text" value=""/>	<input type="checkbox"/> Deck C	<input type="text" value=""/>
<input checked="" type="checkbox"/> In any deck	<input type="text" value="1"/>	<input type="checkbox"/> In each deck	<input type="text" value=""/>	Total	<input type="text" value="1"/>

Back Next

Figure 48: Specification for common modules

7.1.7. Designer specification for templates

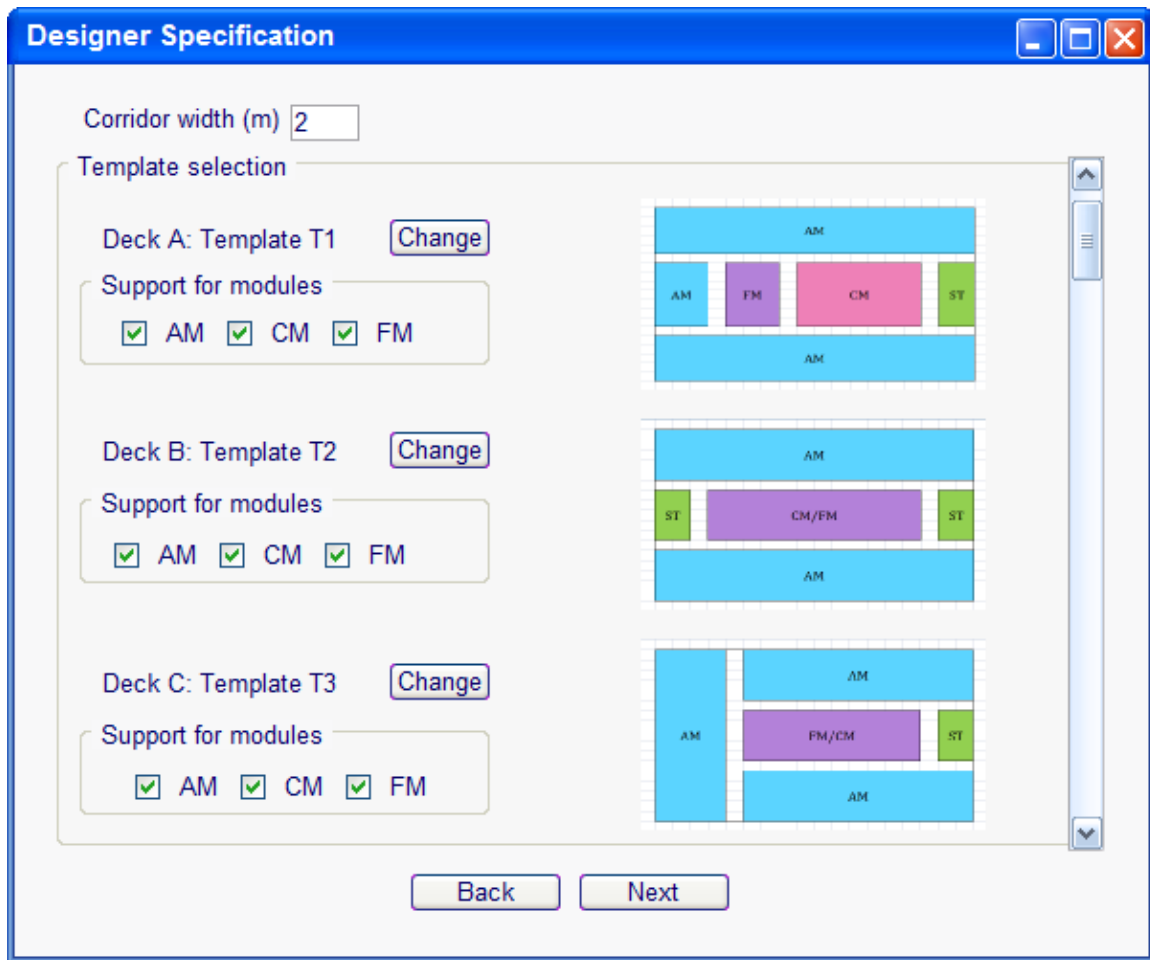


Figure 49: Template suggestion by the system

As previously stated, the system will suggest templates for each deck depending on the input information and vessel database. In this step the designer can override this selection if s/he thinks otherwise. A press on the 'change' button will introduce another window with a list of templates applicable for that deck and the designer is free to choose any one of them for the said deck. The window will also include relevant information for the template such as type of modules that can be placed on the deck, percentage of area for different types of modules and so on.



Figure 50: Template selection by the designer

7.1.8. Module specification according to the database and further adjustment

Required space for common modules will be calculated from the vessel database automatically by the system. It will suggest length, width and area of different common modules by interpolating and extrapolating the data from database and present them in the following window. The designer can modify these suggestions and enter different numbers if s/he likes. The suggested location can also be altered.

If the owner wishes for specific common modules to be placed in a certain deck, this can be accomplished in this step. The designer can enter these specifications here so that the system will allocate space for these modules according to owner's choice, and not follow the database. If the designer opted for "Fixed value" in "Database option" in figure (), then all of the fields in this window will be left blank so that the designer can enter suitable numbers here by choice.

Module specification according to vessel database

Module	Length(m)	Width(m)	Area(sq. m)	Min L/W(m)	Location
Office 1	4	2	8	2	B
Office 2	4	3	12	3	A/B/C
Store 1	3	3	9	3	A
Store 2	5	2	10	2	A
Galley	4	2	8	2	A
Wardrobe	2	1	2	1	A,B,C

Back Customize Next

Figure 51: CM specification by the system

7.2 Processing phase

After entering all the required information, the DSS will process the data and consult the vessel database for relevant information on common modules. An example screenshot of the vessel database is given below where the information is stored in a systematic manner according to the type of vessel and type of accommodation area [Kawser, 2011]. Noise is introduced in the information on the example vessel database with a range of $\pm 10\%$ due to their proprietary nature. Nonetheless, these numbers are a good indication of accommodation cabin specification for already built vessels.

Based on crew number and special requirements, the system will calculate the area required for common modules and at the same time, it will come up with suitable L/W ratio for proper

OSV Areas & Volumes	PSV			AHTS			OSCV		
	PSV 01	PSV 02	PSV 03	AHTS 01	AHTS 02	AHTS 03	OSV 01	OSV 02	OSV 03
	Area [m2]								
October 2011									
MAIN DATA									
ACCOMMODATION	947	1056	1040	1900	2,130	2,070	2,059	2,339	3675
CREW AND CLIENT SPACES	490	600	558	995	1,141	1,162	1,382	1,484	2379
Crew accommodation	245	270	340	400	900	770	896	1,080	1670
Captain Class Suite	45	45	50	62	74	52	60	64	102
Senior Officer Suite	52	106	160	82	136	70	130	134	410
Crew Single	58	11	55	173	75	195	105	110	320
Crew Double	35	54	0	32	412	285	445	520	476
Corridors	45	55	55	70	120	161	160	200	292
Client accommodation	56	67	0	160	0	0	67	140	143
Client(officer class)	28	35	0	44	0	0	48	54	70
Client(crew class)	8	0	0	97	0	0	0	54	44
Corridors	0	0	0	0	0	0	0	0	0
Corridors	12	10	0	50	0	0	10	20	0
Common areas	178	254	146	256	260	299	325	267	567
Dayroom	32	35	50	105	52	36	90	94	125
Mess	44	42	44	91	19	44	85	12	7
Gymnasium	15	13	31	25	101	42	19	0	200
Wardrobe	30	26	25	8	28	107	42	102	40
Public WC	12	2	6	25	40	34	2	37	80
Space behind vertical walls	0	95	0	0	9	34	105	75	81
Corridors	0	0	0	0	0	16	0	20	29
Corridors	18	20	10	35	62	46	75	30	112
Main & service stairs	68	60	72	96	84	120	96	64	116

Figure 52: Vessel database (sample)

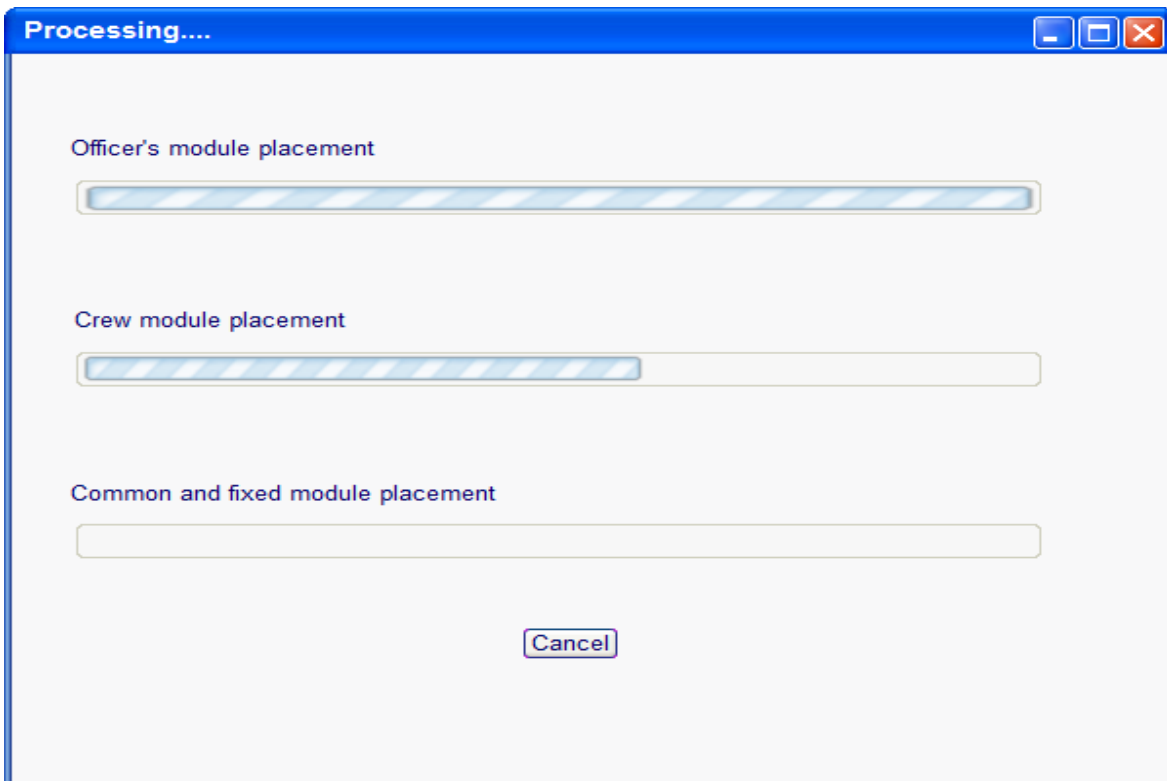


Figure 53: Processing window

orientation of the module. During processing, a screen will show the progress. The user can cancel and go back to the input phase any time s/he wants.

If the selected template is not suitable for proper allocation of space for the vessel, then the system might show some error

messages. In this case the designer might have to change the template or alter some specifications for the relevant modules.

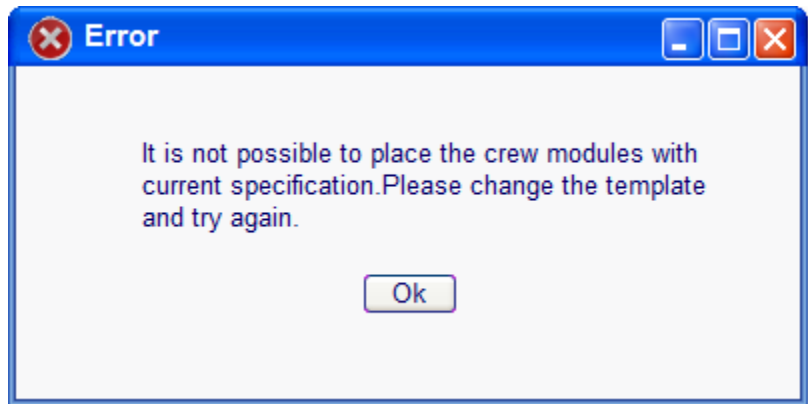


Figure 54: Error message

7.3 Output phase

The DSS will provide output in two steps. First it will show a graphical illustration of the decks containing all the modules at their designated areas. It is an effective way to show the owner how the decks will look like after placing the modules. Although the output will be in 2D format, it will demonstrate the space allocation with color-coded modules for easy interpretation.

Additionally a window will show the details of the decks in text. It will provide the information in tabular format according to the type of modules. Area, dimensions and location of individual module can be readily extracted from this window.

In the following figure, the white area denotes corridors on that deck. The violet area indicates the unused space of the template, which can be utilized by further scaling of the modules in that region by the designer. For common modules, it is also possible to manually drag the borders of the module to decrease/increase their size. Here, the modules CM1 and CM2 can be freely adjusted because of the availability of space around them. If there was no extra space, then this option won't be available at this step.

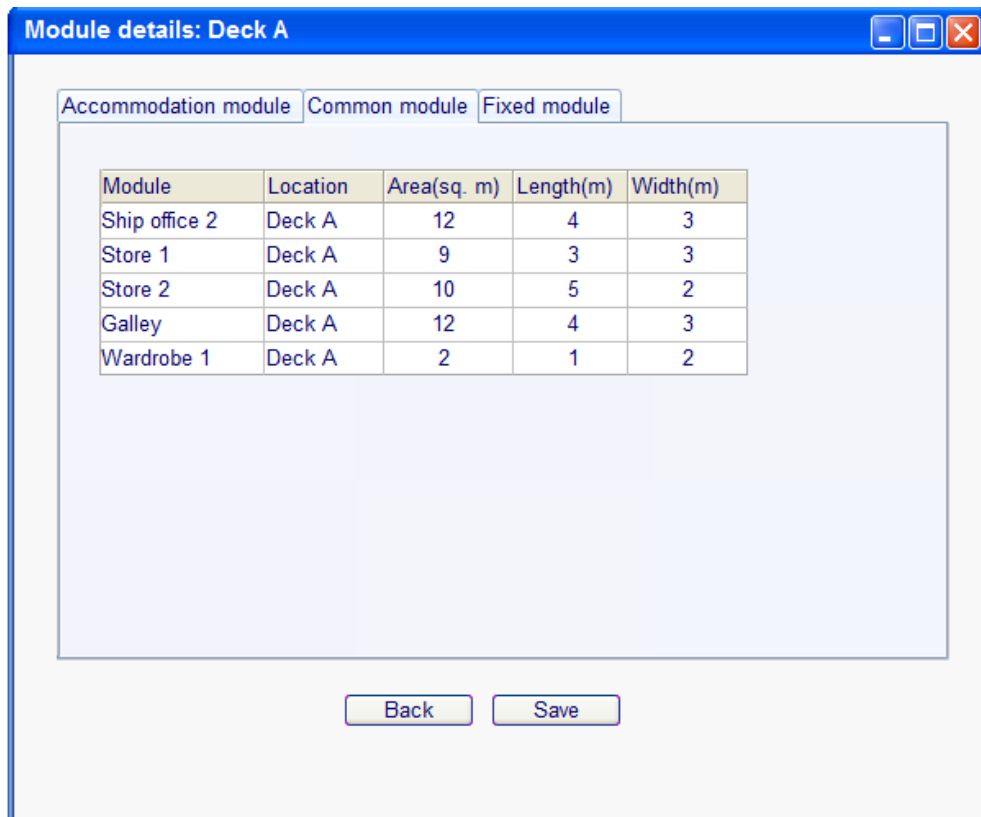
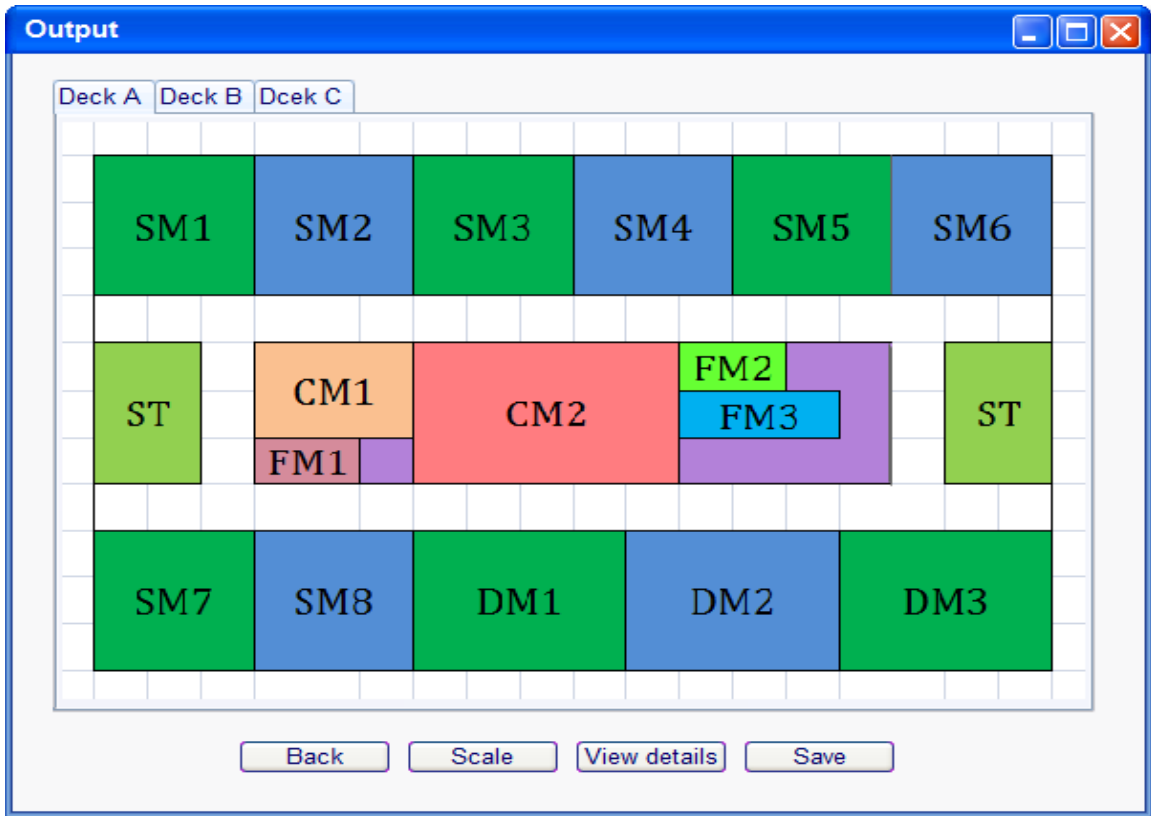


Figure 55: System output

8. Real case study

In order to check the feasibility of the DSS, a real case study is performed in this chapter. This case study is built upon the specification of an AHTS built by STX, AH-11. Since a full-fledged program is not in the scope of this research work, a stripped down program has been written in Java, which covers the bare essentials of the DSS. The numbers used here do not directly reflect the dimensions of the cabins of AH-11. They can be thought of as representative values for a typical AHTS accommodation.

For this DSS, the most versatile object oriented programming language Java (JDK 1.6) is used. It is a suitable choice for developing any Graphical User Interface (GUI) as well as for programming. By using Java it is possible to read input from an Excel file, to process the data efficiently and draw the output as 2D graphics. For 2D graphical output, java swing component *JFrame* is used which gives the window view. A canvas of java abstract window toolkit is used to draw the actual output in the window.

As a simplified study, not all of the characteristics of the DSS have been implemented here. Major limitations are:

- Complete Bi-Partite matching is not implemented here for placing the common/fixed modules. It performs a deck-wise partial/approximate matching.
- In the lower decks the space for CM/FM is not fully utilized because the modules are placed only in a single row. So even if there is plenty of remaining space along the deck width, it will not be used properly and some modules might not get placed.
- The modules are placed from top left side of each template, and gradually goes down from there. Placement of ranks is performed according to rank. It means sometimes the modules might not be placed optimally in order to comply with the rank based arrangement system employed in the system.
- No scaling is implemented, which requires much more expertise and time to program.
- For accommodation modules the dimension close to square size is chosen and placed.
- Because of these lacking, the outputs from the program would not be always optimal.

The programming code can be found in the Appendix. It contains more than 1200 lines of code. The program has been included in a .ZIP file with the thesis submission package. The input file is prepared in Microsoft Excel. The rows are fixed and should not be changed. The (*) sign implies that the module can be placed on any deck as long as there is space available. It reflects the

option 'on any deck' of the DSS when placing modules on decks. If a module has to be placed on a specific deck, it can be done by inserting the deck label on designated row and the size of the module on the right hand side cell.

	A	B	C	D	E	F	G	H
1	No. of deck	4						
2	Deck area, length, width	375	25	15				
3	No. of rank	4						
4	No. of personnel in each rank (1 -> 2 -> ...)	2	3	2	4			
5	No. of crew	38						
6	Single crew module size	9						
7	Double crew module size	20						
8	Officer module size(rank1, rank2, ..)	20	18	16	15			
9	Fixed module specification							
10	Staircase(per deck, size)	2	5					
11	Cleaning locker(per deck, size)	1	5					
12	Public WC	A	5	B	5			
13	Storage space							
14	Engine casing	A	6	B	6	C	6	
15	Common module specification							
16	Mess	A	30					
17	Conference room	*	20					
18	Ship office	*	12	B	20			
19	Gymnasium	*	10					
20	Wardrobe							
21	Stores							
22	Dayroom	*	20					
23	Galley	A	36					
24	Provision spaces	*	12					
25								

Figure 56: Input file for case study

Templates

For this case study, two templates have been used. Since this case is for demonstration purposes only, it does not contain any more templates to choose from. These two templates are fixed in the code by default and the system will try to place modules according to these.

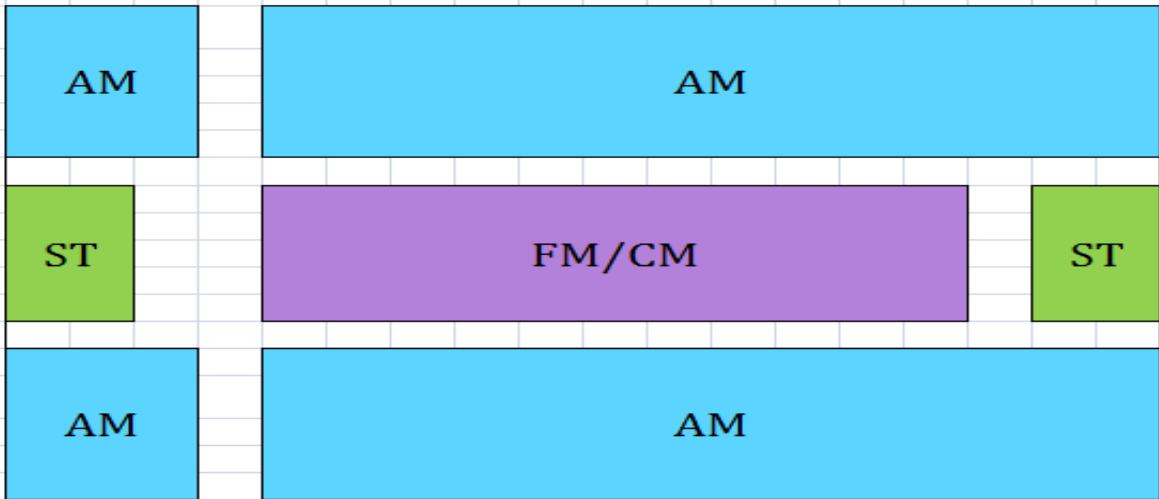


Figure 57: Template for arrangement for the uppermost deck (Deck-D)

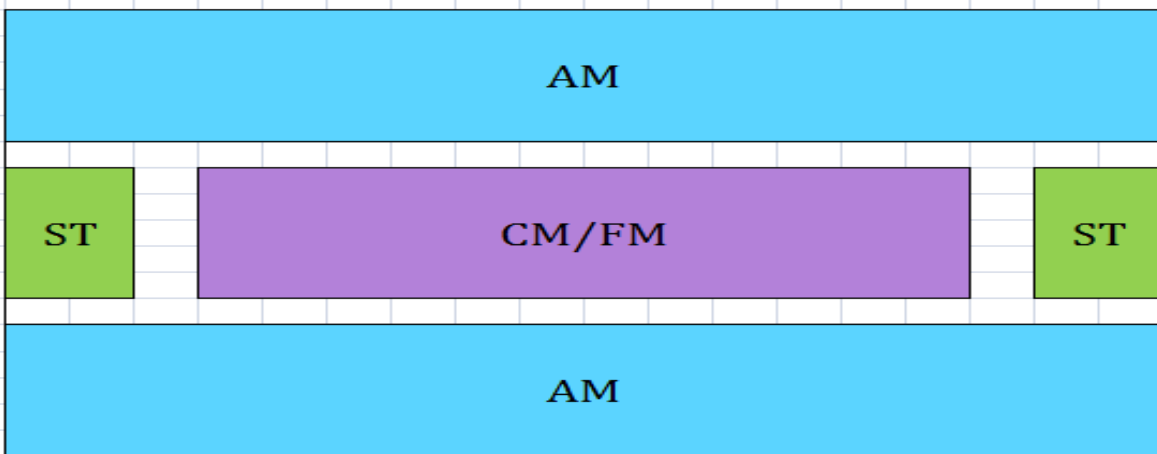


Figure 58: Template for arrangement for lower decks

Results

The results are shown deck-by-deck basis below. The modules are shown in different colors separated by thick black lines. They are not scaled, that is why the arrangement seems asymmetric. Decks are showed in different tabs from upper to lower deck. If it is not possible to place the modules according to the template, then the tool will show an error message, "Placement is not possible for current template. Please change the template and try again".

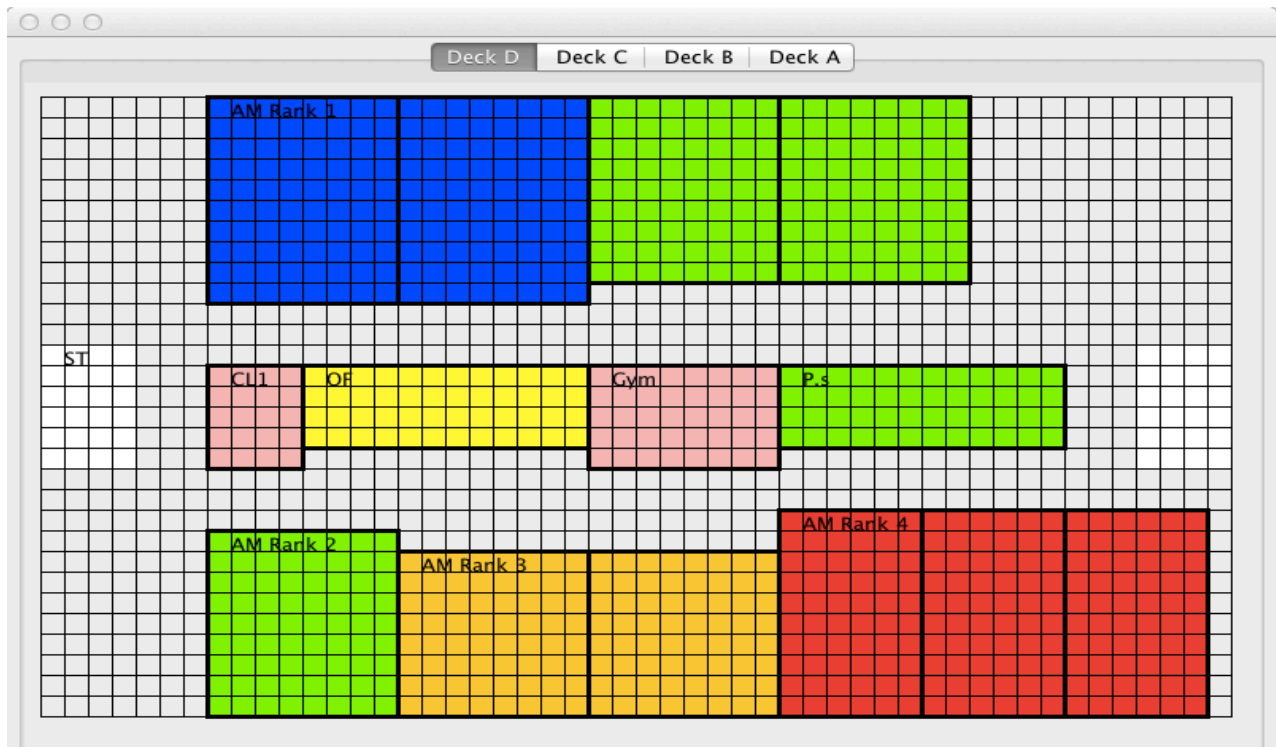


Figure 59: Output Deck D (uppermost)

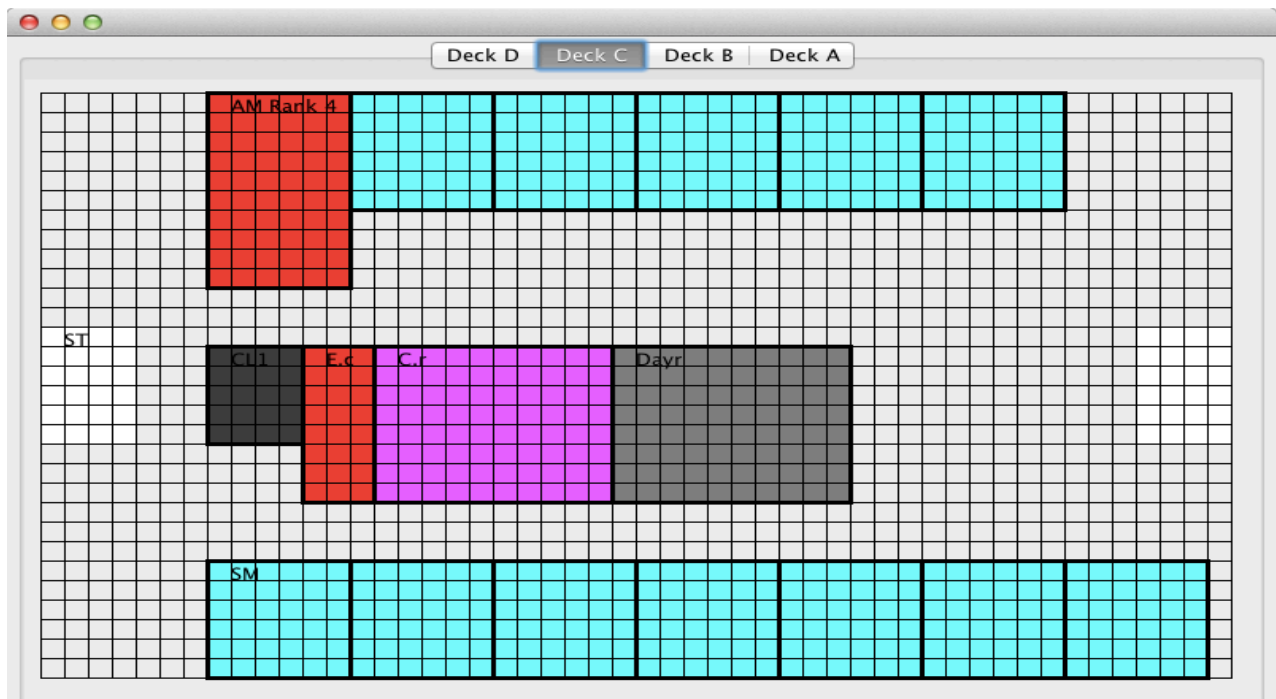


Figure 60: Output Deck C

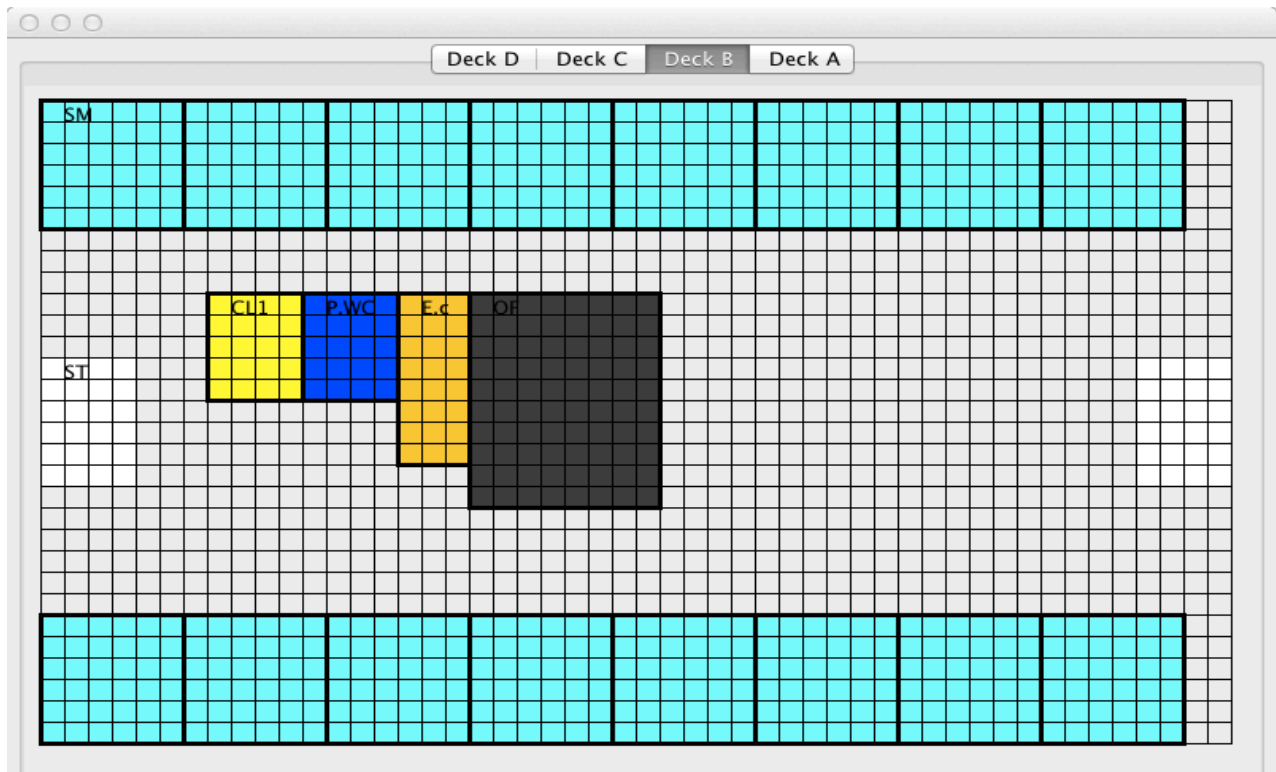


Figure 61: Output Deck B

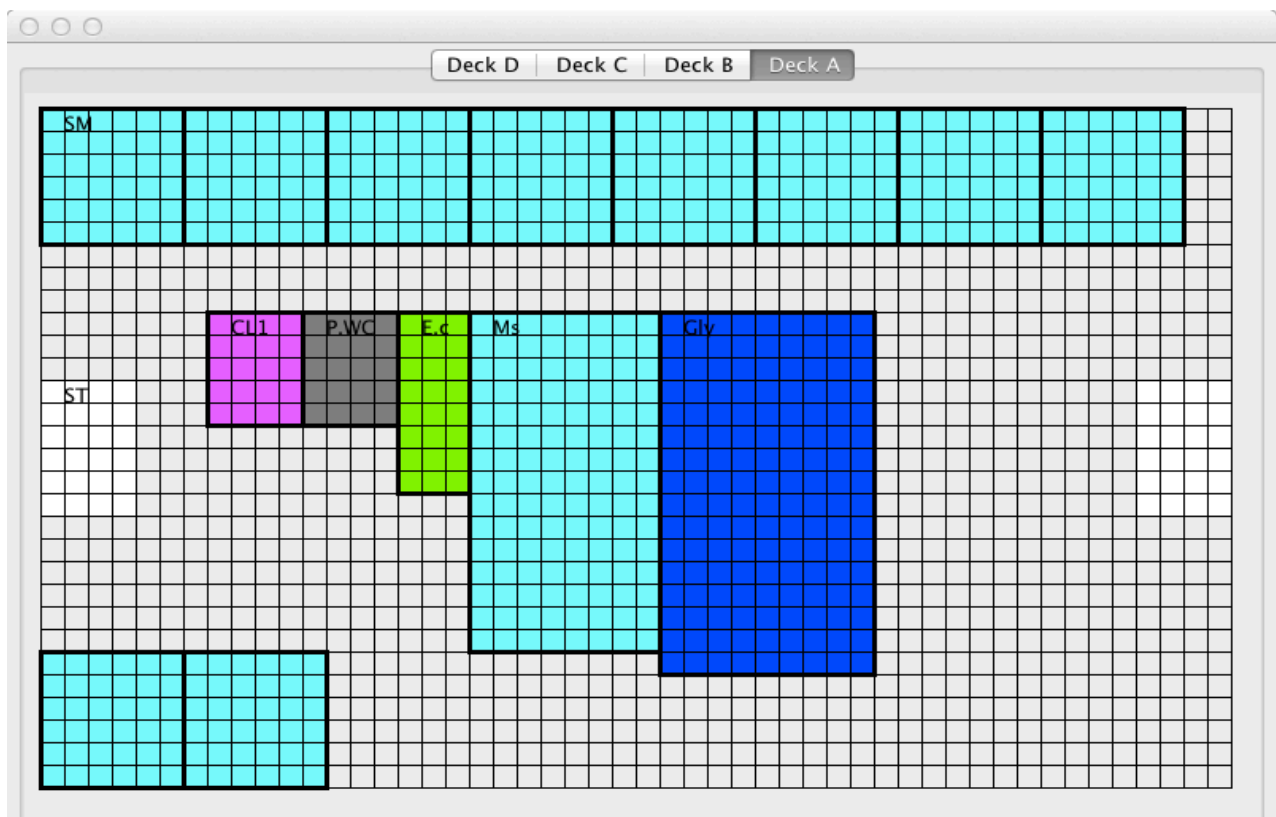


Figure 62: Output Deck A (lowermost)

9. General discussion

9.1 DSS – the rationale

The framework developed in the earlier chapters of this thesis is designed to assist the designer to come up with a fitting accommodation model for an offshore support vessel meeting all the special requirements of the owner and those of the class. The system accomplishes this by optimizing the usage of available space onboard the vessel. It starts with minimum space requirement criteria and goes on from that by allotting space for the required number of modules and places them according to some pre-defined templates. The basic rationale behind this approach can be depicted as follows:

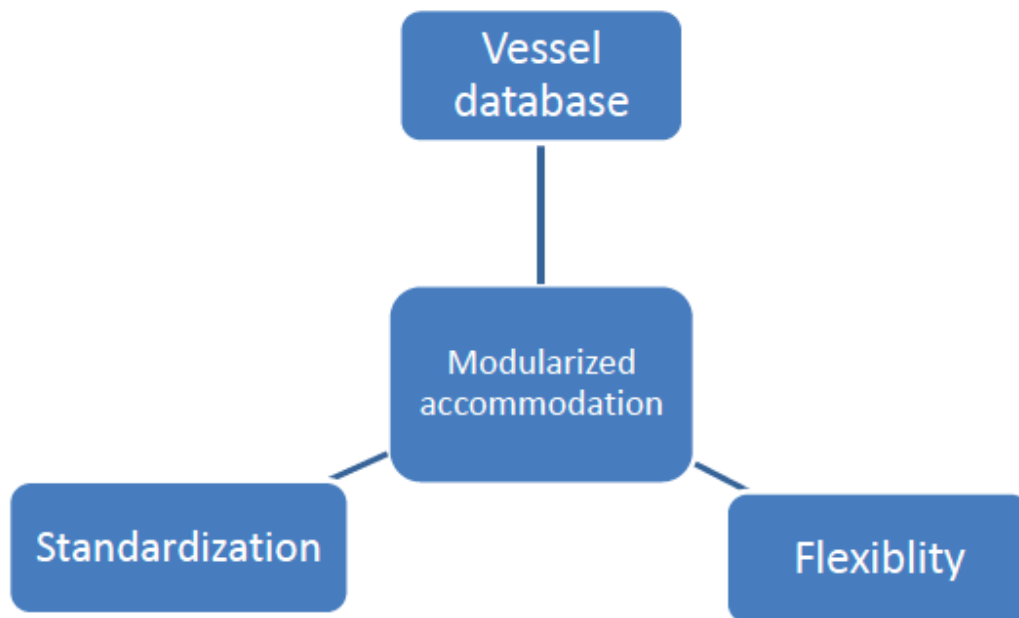


Figure 63: Design rationale for the framework

Standardization is a key feature in modular building methodology. To achieve a modular architecture from a functional one, some characteristics of the problem area must be standardized [Kawser, 2011]. In this case, the fixed modules, crew single and crew double modules are examples of standardized options for the system. These modules will have fixed dimensions so that they can be applied to a range of vessels without much scaling. Though the system will have scaling capabilities for the aforementioned modules just like the others, but the user, depending on the type/scope of design task can restrict that.

One of the most important aspects of the framework is the standardized template library for arrangement of the modules. Arranging the modules over a number of deck levels can be a cumbersome process for any system as there are endless possibilities for arrangement. This is particularly true for large OSCVs or AHTSs where the number of modules can be very high (150 - 250). These templates put boundaries to this voluminous design problem. The system will place the modules according to the selected template for different deck levels, which can be specified by the designer or the system itself. This in essence, lessens the burden of searching for solutions in a vast solution space for possible arrangement. It also helps to speed up the whole process so that the results can be obtained in minimum possible time, which is beneficial for both the designer and the owner/client. Since this framework will act as a preliminary design tool for the organization, timely communication with the stakeholders is vital and this standardized approach helps to achieve that in a pragmatic manner. It is arguable that the templates can limit the design possibilities to some extent and thus eliminate the opportunities for novel vessel design, but this issue can be tackled by introducing more templates from time to time, thus enlarging the design scope of the system.

This framework has its roots partially in another framework for accommodation block design done by the candidate in a pre-thesis project. System based ship design (SBSD) was an integral part of that project. This SBSBD approach has an established vessel database containing design information from a plethora of vessels. This database has the potential to be used in design tools for future offshore vessels as it contains useful information regarding area, volume and location of different spaces of the vessel in a structured manner. In this framework, SBSBD database is used to derive the necessary area requirement for different common modules. This can also be helpful to design templates for vessels since it provides information on deck-by-deck basis. Another important aspect of this implementation is that it will introduce some common features and general trend in newer designs based on empirical information of already built vessels.

The last criterion of the framework is flexibility, which is a prerequisite characteristic for any design tool for designing offshore vessels. This particular market segment is notorious for the requirement of highly customized vessels. So, standardizing everything would not be a fitting idea in this regard. There should be enough flexibility in the system that it could satisfy a diverse array of customer needs and do it in a modern and systematic way [Kawser, 2011]. Flexibility is implemented in the system in various ways:

- The template library can be upgraded with new templates that will enable the system to stay up to date.

- When designing a vessel, the designer is free to choose any template from the library where the option for choosing different templates for different deck levels is also supported. More over, the existing templates can also be modified by the designer for particular vessels if the need be.
- The designer can accommodate special design requirements of the owner in each design step. For example, if the owner needs certain modules to be placed on some preferred location of certain deck level, and then this can be accomplished by simply specifying the need in the template selection step of the system.
- The system is designed to deliver results in real time. If the owner or the designer is not satisfied with the outcome, then they can simply modify the input data or some parameters of design to get a different result. All this can be performed in a simple window based user interface that ensures adequate flexibility for the user.

9.2 Comparison with similar approaches

Because of the nature of the framework, a direct comparison with other similar type of approaches namely Van Oers' optimization based space allocation method (2007) and Andrews' building block design (2003) can be useful. Both of these approaches were built upon different strategies of arranging modules (spaces) for sea-going vessels. These methods are already outlined in chapter 3 of this thesis. Here, comparisons with the developed framework will be made in details.

9.2.1. Van Oers approach

This approach is a completely automated one that can be applicable for not only the accommodation block, rather the whole vessel. Although the author has developed this method with naval vessels in mind, nevertheless it is suitable for other types of vessels as well.

One major difference of this approach from the framework is that the scope of positioning of the modules. As the positions of the modules are more limited in this framework because of the templates, the solution space gets restricted and results can be achieved much faster. The choice of programming language dictates the time required for generating outputs as well. Matlab is currently used by Van Oers for his packing approach which is a slower alternative compared to Java used in this DSS. Van Oers used NSGA-II genetic algorithm for locating the feasible solutions, whereas this DSS employs simple optimization algorithm for best solution. One advantage of

such an advanced algorithm used by van Oers is that it can be helpful to compare among different feasible solutions with varying system choices, configurations and performance levels [Van Oers, 2011]. So essentially it is easy to identify the trade-offs and the consequences of design choices and filter down the alternatives to be settled on a final design. The DSS developed here is primitive in the sense that it will give single output depending on the user input according to the system constraints and rules. Comparing the results in this DSS is not as straightforward as it is in Van Oers method since there are no performance levels/MOMs or other criteria based on which one can distinguish among alternatives. Nevertheless, the user can generate as many as outputs as s/he wants by altering the input parameters to get an idea of impact of design choices on the vessel.

One advantage of the DSS over Van Oers method is the simplicity the templates and the smaller sized modules provide for arrangement of spaces. Van Oers had to implement algorithms for detecting overlapping of spaces and effective ways to manage these overlaps. Sometimes the objects/spaces that has to be located at a certain deck level can be very large and splitting of objects is necessary for arrangement. But splitting requires additional algorithm [Van Oers, 2011], which is not quite developed yet. Because of the smaller size of the modules and the fact that size of the crew single/double and officers modules are fixed, it is easy to allocate space for them in this DSS without the need for splitting. The templates play an important role here to minimize overlapping.

9.2.2. Design building block approach

Focusing on the basic functions that a ship has to perform, David Andrews and his colleagues have published a number of papers on 'design building block approach', which is described in chapter 3. This University College London (UCL) approach combines numerical and spatial descriptions in a single integrated approach to create ship designs that are feasible. In this sense, the developed DSS is similar to the design building block approach since it also contains numerical and spatial descriptions in a unified manner. Although the systems by which this information is stored are completely different for these two approaches, the design rationale is almost the same, which emphasizes human responsibility for design generation to create satisfying results at the cost of significantly reducing the number of alternatives considered. The templates used in the DSS constrict the feasible solution space to a much narrower scope, which also involves greater designer intervention in different steps.

One major difference between these methods is the implementation of optimization. In Andrews' building block method, ship concepts are validated and not necessarily optimized for real world application. The user makes improvements by studying the effect of each block on the design's performance by moving the block and re-evaluating the analysis. Andrews proposed that the goal of preliminary design was specification development, rather than development of the design itself [Nick, 2008]. On the other hand, the DSS developed in this research work starts with the specifications as input and builds the design from there. It utilizes optimization techniques for solving real world issues and can be readily implemented in new vessel designs. Although, the primary goal of the DSS is to assist the designer and at the same time demonstrate the overall arrangement in a visually simple and illustrative manner to the client, nonetheless it has the potential to be applied in future vessel designs, not only preliminary but also in detailed phase.

Another noteworthy difference between these two approaches is the notion of novel design. In Andrews' building block method, innovation and exploration of design space are unrestricted in most of the cases. The design procedure commences with a very broad outline of the new warship. The building blocks are packed first, before an envelope is wrapped around the set of objects. It can be thought of as a way of considering the possible solution as lying in a solution space, which, at the simplest level, could have three 'dimensions' of packaging, technology and capability. The approach is capable of being both exploratory and divergent, so that the design team would be able to explore and be informed by possible good ideas [Andrews 1998]. On the contrary, the DSS in this research work does not incorporate innovation as a basis of design. Rather it is more like a routine procedure, which depends on empirical data and searches in a restricted design space dictated by templates and rules from class. New technology or investigation of new concepts is not the prime application for the DSS.

Overall, Andrews' building block approach is suitable for more challenging and open conceptual design problem where innovation can play a vital role. Again, the DSS developed here will produce its best results when the specification is pre-defined and the design space is well formulated with structured information.

9.3 DSS – Known issues

Despite the fair amount of efforts spent on the development of the DSS, there are apparent shortcomings of the system. This DSS has been developed at a relatively high level of detailing,

where not all of the lower level details are addressed. Here, these imperfections will be discussed in the following list.

- The system starts with an assumption that the area of each deck of the vessel under design is known. In some cases, these values can be unknown to the designer. When designing a novel vessel or a vessel with unusual proportions, it may pose as a challenge to predict the area available on each deck. The vessel database would not be of much help at these cases. Hence, the DSS may not be much useful to the designer under these circumstances.
- The system is unable to handle modules that are not rectangular. It is a major drawback of the DSS. Most vessels have curvilinear shape at the front where the accommodation block is usually placed. As a result, the modules located on the curvilinear portion of the hull may have sides that are not straight lines. Hence, the need of non-rectangular modules comes into play. Managing modules of different shapes will be a considerable hurdle for any DSS of this nature.
- Templates are used in the DSS to facilitate easier allocation of space for the modules. As with everything in this earth, this simple method has its flipside as well. Employing templates will make it relatively easy to establish a computer tool based on the DSS, but at the same time it will limit the scope of design by the system. Since the system will only look into the available template library that might not be sufficient for all kinds of vessels; it may come up with a solution that can be shortsighted or impractical. Although this issue can be largely addressed by adding more templates from time and time and/or update the existing ones whenever necessary. The fact that the system can be upgraded by adding more template modules in future can be thought of as a compromise between easier implementation and design flexibility.
- The lack of measures of merit (MOM) is another issue of this DSS. MOMs are general terms for all measures that characterize a system under analysis. Technical measurement is important and MOMs are the criteria by which a system can be evaluated whether it is going to meet the objective set by the user. For a system like this, MOMs can be the maximum distance of exit from modules, accessibility to modules from different sides and so on. These can be very useful to compare different alternatives and choose the one which best fits the designer's intentions.
- In some instances, there may be too much space available for the modules. The system will try to scale the modules to utilize this extra space. But such scaling may result in different sizes of the same module in different deck levels, which may not be desirable by

the designer. Scaling is done in pre-defined steps; which may not be an efficient way to utilize space at all times.

- The output of the DSS is in 2D format. The designer/owner will get a color-coded 2D layout of the allocation and placement of modules on different decks. The output is informative but may not be the most eye-catching one in this regard. Now a day, more and more design tools are developed to generate results in 3D format that is more visually attractive and easier to interpret. Since, this tool will work as a sales pitch for the potential owner/client as well, 3D should be the way to go.

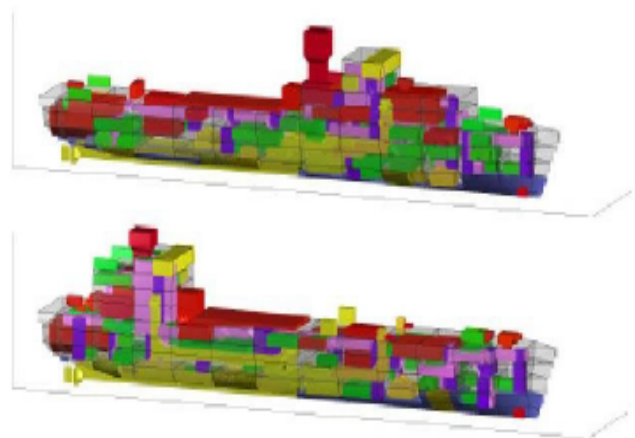
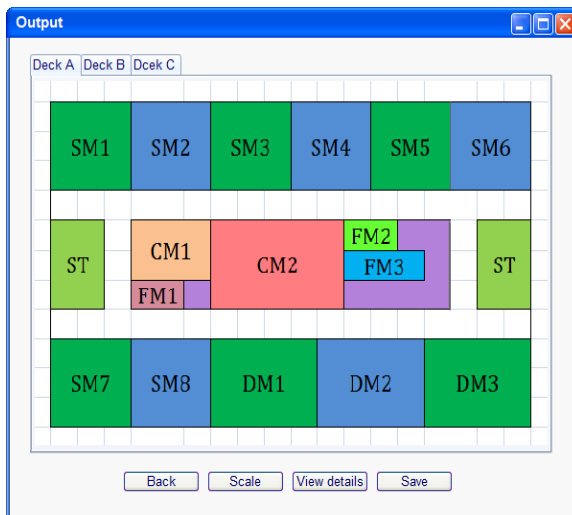


Figure 64: 3D output from Van Oers approach compared with the output from DSS

- The current computer program, which is developed for the real case study is a bit slow for rapid generation of results based on changing input data. In real world, the processing time should be faster for effective communication with the client.
- Nothing has been done here with regard to cost or production-friendliness. Cost can be a determining factor when evaluating modular concepts. After all, implementation of modularization would not be feasible if the price/cost of production does not go down because of it. Although most of the studies regarding modularization indicate that lower price is an advantageous outcome of modular production, for a complex industry like OSVs, things can turn out to be different.
- This research has been totally focused on the accommodation block only. It might be a good idea to evaluate the impact of modular accommodation to other parts of the vessel. If a combined production strategy is used where only the accommodation is built by modules and the other sections in the conventional way, integration issues may arise.

9.4 Further development

The algorithm developed in this DSS is intended to outline a more detailed approach for compiling a tool for OSV accommodation design. Hence, much work is needed to make it a full-fledged computer tool. It is also important to note that tools like this that are designed to assist the designer at an early stage of development must be fast enough to provide rapid output for active communication with the owner/client. The algorithm needs more structuring and articulation in this regard. Based on the aforementioned issues regarding the DSS, the following steps can be taken to rectify the shortcomings.

- The ability to handle modules of different shapes (non-rectangular) and sizes should be introduced in the system. This may seem as a far-fetched vision for the current system. But small steps can be taken at a time to continually improve the flexibility of the system in future. For instance, some of the modules being designed by STX OSV are hexagonal. So for the next logical step, it may be a good idea to try to implement modules with five sides and so on.
- The output from the system should be in 3D, which will make the visualization of the results more effective. For this, the tool can be linked to third party visualization software like Google Sketchup. Øyvind Vestbøstad (2011) has already shown a way to do this by some clever scripting mechanism in Java and Visual Basic. This kind of design sketching has been used in the early stages of ship design, both as an aid to the development of the design itself and as a communications medium [Pawling et al, 2011].
- MOMs should be implemented in the system for evaluation. Without them, it is not easy to say whether an arrangement is optimal or not. It may very well satisfy the requirements laid out by the client, but there might be better ways to solve the design issues. Quantification of MOM can be a difficult task to accomplish.
- The algorithm should be streamlined to minimize the processing time. As said earlier, effective communication with the stakeholders needs brisk software that can take inputs and generate results in real time. Long processing time would hinder the usability of this DSS as an assisting tool.
- Automated data mining from web sources can be introduced which will complement the vessel database from SBSDB. Classification societies like DNV, LR store data from a plethora of vessels in their database that can be valuable for a DSS like this. The only pitfall is that the data in these databases may not be properly structured for direct application in this system. Gathering data from various sources that are stored in different formats is not an easy task to accomplish. It will need considerably powerful data mining tool. Nevertheless, it will be worthwhile when the design tool is working properly.

10. Conclusion

Offshore shipbuilding industry is traditionally a conservative one where new ideas or modern methods of designing are difficult to implement. But the need of standardization is not an issue that can be overlooked anymore. Accommodation block of offshore vessels is a good starting point for practicing modular building methodology in this industry. Not all of the spaces in the accommodation block should be modularized to strike a fine balance between standardization and customizability. Application of MFDTM method suggests that crew single, crew double and officer's cabins are the prime candidates for a modular product platform.

A DSS has been developed to assist the designers to design the accommodation block in a systematic manner. Based on standardized modules and vessel database, it will come up with a layout for accommodation spaces satisfying the customer requirements. A mock-up of a user interface of a computer tool built on the DSS has also been realized. The feasibility of the DSS has been tested with an illustrative example and later with a real case study. The results from the case study are promising for further development of the tool. By refining the algorithm and implementing a 3D output method for the DSS, it has the potential to be used in real world designing challenges.

Establishing modular product architecture in offshore shipbuilding industry is by no means an easy task to accomplish. Application of modular thinking for designing the accommodation block can lead the way to a wholly modularized shipbuilding method. The methodology developed here has showed that modularization can be implemented which will introduce standardization in the product platform and at the same time has room for flexibility to accommodate diverse customer requirements.

11. Bibliography

- Andrews, D.J.**, "An integrated approach to ship synthesis", Transactions of the Royal Institution of Naval Architects, 1986
- Andrews, D.J.**, "A comprehensive methodology for the design of ships", Proceedings of the Royal Society, vol. 454, 1997
- Andrews, D.J.**, "Simulation and the design building block approach in the design of ships and other complex systems", Proceedings of the Royal Society, vol. 462, 2006
- Andrews, D. J., Burger, D. & Zhang, J.**, "Design for production using the design building block approach", RINA International Journal of Maritime Engineering, 2005
- Blecker, T., Abdelkafi, N., Kreuter, G., Friedrich, G.**, "Product configuration systems: state-of-the-art, conceptualization and extensions", MCSEAI Tunisia, 2004
- Bohanec, M.**, "What is decision support?", Proceedings from Information Society IS-2001: Data Mining and Decision Support in Action!, 2001
- Brathaug, T.**, "Product configuration in ship design", Master thesis, Department of Marine Technology, NTNU, 2008
- Brathaug, T., Holan, J. O., Erikstad S. O.**, "Representing design knowledge in configuration-based conceptual ship design" COMPIT, Liege, 2008.
- Chandrasekaran, B., Stone, R.B., McAdams, D.A.**, "Developing design templates for product platform focused design", Journal of Engineering Design, vol. 15, 2004
- Christiansen, K., Tvenge, K.**, "Shaping product portfolios for the future", Valcon management consultants, [Report], 2003
- CS787:** Advanced algorithm lecture notes, University of Wisconsin, [Online], [<http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec5.pdf>], Accessed March 27, 2012
- Dahmus, J.B., Otto, K.N., Gonzalez-Zugasti, J.P.**, "Modular product architecture", Design Studies, vol. 22, issue 5, 2001
- Eggen, O.**, "Modular product development: a review of modularization objectives as well as techniques for identifying modular product architectures, presented in a unified model", Department of Product Design, NTNU, 2003
- enge.vt.edu** website, [Online], [http://www.enge.vt.edu/terpenny/Smart/Virtual_econ/Module2/pahl_and_beitz_method.htm], Accessed February 20, 2012
- Ericsson, A., Erixon G.**, "Controlling design variants: modular product platforms", Society of Manufacturing Engineers, ISBN 978-0872635142, 1999

Erikstad, S.O., “Modularization in shipbuilding and modular production”, Innovation in Global Maritime Production – 2020 [Working paper], NTNU, 2009

Erikstad, S.O., Lecture notes on TMR 4115 [Design methods], Department of Marine Technology, NTNU, Fall 2010

Erikstad, S.O., Hagen, A., “Applying product platform technologies in ship specification development”, International Marine Design Conference [IMDC], MI, 2006

Erixon, G., Kenger, P., “Development of modular products”, Proceedings of 4th student conference on modular products, Dalarna University, Sweden, 2006

Gockowski K., “Modularization in ship equipment”, Intermodul s/03/G, [Report], October 2005

Hölttä-Otto, K., “Modular product platform design”, Doctoral dissertation, Department of Mechanical Engineering, Helsinki University of Technology, 2005

Huang, C.C., “Overview of modular product development”, Proceedings of the National Science Council, Republic of China, Part A (Physical Science and Engineering), 24(3), 2000

Katsoufis, G.P., “A decision making framework for cruise ship design”, Master thesis, Department of Mechanical Engineering, MIT, 2006

Kawser, Z., Report on summer internship on Ship4C project, [Report], August 2011

Kawser, Z., “Modularization and configuration based design of offshore support vessel accommodation areas”, Department of Marine Technology, NTNU, [Report], 2011

Kenger, P., Lecture notes on “Development of Modular Products”, Dalarna University, Sweden, 2006

Kidd, P.T., “Agile manufacturing: a strategy for the 21st century”, [Report]

Kreng, V.B., Lee, T.P., “QFD-based modular product design with linear integer programming—a case study”, Journal of engineering design, June 2004

Kusiak, A., “Concurrent engineering: automation, tools and techniques”, Wiley-Interscience, ISBN 978-0471554929, 1992

Kusiak, A., “Integrated product and process design: a modularity perspective”, Journal of engineering design, 13(3): 223-231, 2002

Lean management instituut, “Volvo learns to use Modular Function Deployment”, [Report]

Levander, K., STX Europe company presentations, STX Europe, Finland

Levander, K., “System based ship design”, NTNU, 2009

Liker, J.K., Lamb, T., “What is lean ship construction and repair?”, Journal of ship production, vol. 18, issue 3, August 2002

Lu, N., Korman, T., “Implementation of Building Information Modeling (BIM) in modular construction: benefits and challenges”, Construction research congress, American Society of Civil Engineers, 2010

Mistree, F., Smith, W.F., Muster, D., Allen, J.K., “Decision-based design: a contemporary paradigm for ship design”, Proceedings of SNAME, 1990

Moyst, H., Das, B., “Factors affecting ship design and construction lead-time and cost”, Journal of ship production, vol. 21, issue 3, August 2005

Mullens, M.A., “Production flow and shop floor control: structuring the modular factory for custom homebuilding”, Proceedings of National Housing Research Agenda for Construction and Production, ASCE, 2004

Nepal, B.P., “An integrated framework for modular product architecture”, Doctoral dissertation, Wayne State University, 2005

Nick, E.K., “Fuzzy optimal allocation and arrangement of spaces in naval surface ship design”, Doctoral dissertation, University of Michigan, 2008

Pahl, G., Beitz, W., “Engineering design: a systematic approach”, Springer, ISBN 9781846283185, 1999

Pawling, R., Andrews D., “Design sketching – the next advance in computer aided preliminary ship design?”, COMPIT, Berlin, Germany, 2011

Pimmler, T.U., Eppinger, S.D., “Integration analysis of product decompositions”, ASME Design Theory and Methodology Conference, Minneapolis, 1994

Rode, V., Industrial designer, STX OSV Design AS

Simpson, T.W., “Product platform design and customization: status and promise”, DETC, Ai Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing, 18(1): 3-20., 2004

SmartMarket Report: “Prefabrication and modularization: increasing productivity in construction industry”, McGraw Hill Construction, [Report], 2011

SRI online, Maths & Decision Systems Group, Silsoe Research Institute, [Online], [<http://www.sri.bbsrc.ac.uk/scigrps/sg9.htm>], Accessed February 22, 2012

Stone, R.B., Wood, K.L., “Development of a functional basis for design”, Journal of mechanical design, 2000

STX OSV official website, [Online], [<http://www.stxosv.com/Pages/default.aspx>], Accessed March 14, 2012

Tsai, Y.T., Wang, K.S., “The development of modular-based design in considering technology complexity”, European journal of operational research, vol. 119, 1999

Ulrich, K., Eppinger, S. “Product design and development” 4th ed., McGraw-Hill, 2007

Ulrich, K., Tung, K., “Fundamentals of Product Modularity,” Issues in Design Manufacture/Integration, ASME, 1991

van Oers, B., “A packing approach for the early stage design of service vessels”, VSSD, ISBN 978-90-6562-283-9, 2011

van Oers, B., Stapersma D., Hopman J., “Issues when selecting naval ship configurations from a Pareto-optimal set”, AIAA, Victoria, Canada, September 2008

van Oers, B., Stapersma, D., Hopman, J., “A 3d packing approach for the early stage configuration design of ships”, INEC, UK, May 2010

Vestbøstad, Ø., Master thesis on system based ship design, Department of Marine Technology, NTNU, 2011

12. Appendix

A. System Based Ship Design (SBSD)¹

The SBSB approach was developed for ships by Kai Levander. It has similarities to both expert systems and case based design approaches, with focus on experience data and decision support and can probably best be described as a variant of an expert system.

The model has been developed for container vessels, RORO vessels and cruise vessels giving estimates of new builds based on earlier designed vessels. These vessels are to a large extent generic, and follow a pattern in the design with small differences thus the design task is more of a scaling issue. To estimate the need for displacement the method uses a bottom up strategy to determine the needed area, space and weight for each sub function of the new build, and thereby estimates the displacement, main dimensions and building costs.

A functional breakdown is used to be able to utilize the outcome for statistical purposes for new vessel projects. Andrews, Pawling, Casarosa, Galea, Deere, Lawrence, Gwynne and Boxall distinguish the main functions of naval vessels in “Float”, “Move”, “Fight” and “Infrastructure” while Levander (2009) firstly split the vessel into the categories “Ship functions” and “Payload functions” where the “Ship Functions” are functions that are needed to operate the ship, independent of the cargo on board. The Payload Functions are functions and requirements that generate cash flow for the vessel. For instance it is a requirement for chemical tankers to have the ability to heat their cargo while the main function of the payload is the cargo space. For transport vessels with limited variations of products, it is normally easy to distinguish between Ship Functions and Payload Functions since the systems are more or less split. For offshore vessels this is a lot more challenging.

Currently, through the Ship4C project Kai Levander, NTNU and STX OSV are exploring how the approach could be deployed for offshore vessels such as PSV, AHTS and OSCVs. Levander has developed a new function hierarchy designed specifically for OSVs. Instead of the term Payload Functions, he uses Task Related Functions, which is more wide-ranging and fits specialized vessels better. In this category all cargo related functions are added, as well as service related functions such as anchor handling winch, offshore crane and so on.

¹ This excerpt is taken from the master thesis of Øyvind Vestbøstad (2011)

By calculating without drawing, the SBSB approach does not lock assumptions in the concept phase and will thereby support a more creative process in the start of the project. Levander describes SBSB as:

“System Based Ship Design is like a checklist that reminds the designer of all the factors that affect the design and record his choices. It gives the possibility to compare the selections with statistical data derived from existing, successful designs.” (Levander, 2009)

The SBSB method is suitable for early design decisions, and more of a tool to find the best assumptions before starting designing the vessel. The use of SBSB would secure that the new design is based on the most fitted basis ship, and thereby save iterations in the design spiral later on.

B. Code for the real case study

Main class

```
package dssupd;

import javax.swing.*;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/* @author Choudhury */
public class Main{
    public static String moduleNames[] = new String[200];

    public static boolean flag = false;
    public static deckTabs drawDecks;
    public static boolean impossible = false;
    public static int noOfDeck, noOfRank, noOfCrew, deckArea, singleCrewModuleSize,
        doubleCrewModuleSize, corridor, noOfSingleCrewModule, noOfdoubleCrewModule,
        noOfStaircase, areaOfStaircase, noOfModules, deckLen, deckWidth,
        noOfOfficerModules, unplaced, placed, currentDeck, currentRank, rest,
        startX, startY, subGridBreadth, currentPart;
    public static int[] noOfPersonInEachRank = new int[20];
    public static int[] officerModuleSizes = new int[20];
    public static int[][][] decks = new int[10][80][80];
    public static int[][][] c = new int[10][80][80];
    public static module[] allModules = new module[100];
    public static fixedModule staircase, singleCrew, doubleCrew;
    public static fixedModule[] officerModules = new fixedModule[20];

    public static subGrids grid[] = new subGrids[20];

    public static rect[][] rectX = new rect[10][200];
    public static rect[][] rectY = new rect[10][200];
    public static int[] noOfRX = new int[10];
    public static int[] noOfRY = new int[10];

    public static int[][] deckVsMod = new int[10][100];
    public static int[] counts = new int[10];
    public static boolean[] placedMod = new boolean[200];
```

```

public static int modL, modW, minLen;
public static module tm;
public static int extra = 0;
public static String un[] = {"P.WC", "St.s", "E.c", "", "Ms", "C.r", "OF", "Gym", "W", "Sto", "Dayr", "Gly",
"P.s"};

public static void main(String[] args) throws Exception
{
    try{
        String filename = "dss_input.xls";
        List sheetData = new ArrayList();
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(filename);
            HSSFWorkbook workbook = new HSSFWorkbook(fis);
            HSSFSheet sheet = workbook.getSheetAt(0);

            Iterator rows = sheet.rowIterator();
            while (rows.hasNext())
            {
                HSSFRow row = (HSSFRow) rows.next();
                Iterator cells = row.cellIterator();
                List data = new ArrayList();
                while (cells.hasNext())
                {
                    HSSFCell cell = (HSSFCell) cells.next();
                    data.add(cell);
                }
                sheetData.add(data);
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Input file dss_input.xls is not found");
        }
        finally
        {
            if (fis != null) fis.close();
        }
        for(int i=0; i<200; i++) placedMod[i] = false;
        for(int i=0; i<noOfDeck; i++) counts[i] = 0;
        readExcelData(sheetData);

        corridor = 3; //1.5 meter
        for(int i = 0; i<noOfDeck; i++)
            for(int j=0; j<deckWidth; j++)
                for (int k=0; k< deckLen; k++)

```

```

        {
            decks[i][j][k] = 0;
            c[i][j][k] = 0;
        }
    placeStairs();

    for(int pp = 0; pp < noOfDeck; pp++ )
    {
        noOfRX[pp] = 0;
        noOfRY[pp] = 0;
    }

    unplaced = noOfOfficerModules; // total officer module
    placed = 0; // current rank officer module
    currentDeck = 0; // topmost deck
    currentRank = 0; // highest rank
    currentPart = 1;
    rest = (deckWidth - staircase.width - 2*corridor)/2;
    startX = startY = 0;
    int kk = 0;
    while (unplaced > 0 && kk < 2 && kk < noOfDeck)
    {
        subGridBreadth = staircase.length; // for left/right portion
        placeAM1();
        if(unplaced > 0)currentDeck++;
        kk++;
    }
    while (unplaced > 0 && kk < noOfDeck)
    {
        placeAM2();
        if(unplaced > 0)currentDeck++;
        kk++;
    }
    //printDecks(); //debug purpose
    System.out.println("before.....cd = " + currentDeck + " cp = " + currentPart + " cr = " +
currentRank + " rest = " + rest + " dl = " + deckLen);
    System.out.println("before.....up = " + unplaced);

    extra += unplaced;
    //crewmodule processing
    int count = 0; //no of subparts
    int L = 0;
    int w = (deckWidth - staircase.width - 2*corridor) / 2;
    int usableL;
    if(w >= singleCrew.width)
        usableL = deckLen;
    else usableL = deckLen - 2 * (staircase.length + corridor);

```

```

L = 2 * usableL * (noOfDeck - currentDeck -1);
count += 2*(noOfDeck - currentDeck -1);
System.out.println("1.....>>>>.....>>>>>> L = "+L);
if(currentPart == 1) //upper left
{
    if(rest >= singleCrew.width)
    {
        L += rest;
        count++;
        System.out.println("2.....>>>>.....>>>>>> L = "+L);
    }
    else if(w >= singleCrew.width)
    {
        currentPart = 2;
        startX = deckLen - w;
        startY = 0;
        rest = w;
        L += w;
        L += 2*(deckLen - max(max(staircase.length, singleCrew.length)+corridor, grid[currentDeck].left));
        System.out.println("3.....>>>>.....>>>>>> L = "+L);
        count += 3;
    }
    else
    {
        currentPart = 3;
        startX = 0;
        startY = grid[currentDeck].left;
        L += 2 * (deckLen - grid[currentDeck].left - staircase.length - corridor);
        count += 2;
        rest = deckLen - grid[currentDeck].left - staircase.length - corridor ;
        subGridBreadth = 0;
        System.out.println("4.....>>>>.....>>>>>> L = "+L);
    }
}
else if(currentPart == 2) //lower left
{
    if(rest >= max(singleCrew.width, doubleCrew.width))
    {
        L += rest;
        L += 2*(deckLen - max(max(staircase.length, singleCrew.length)+corridor, grid[currentDeck].left));
        count++;
        System.out.println("5.....>>>>.....>>>>>> L = "+L);
    }
    else
    {
        currentPart = 3;
        startX = startY = 0;
        int pl = deckLen - grid[currentDeck].left;

```

```

    if(w < max(singleCrew.width, doubleCrew.width))
    {
        pl -= (staircase.width + corridor);
        startY = grid[currentDeck].left;
    }
    L += 2 * pl;
    rest = pl;
    subGridBreadth = 0;
    System.out.println("6.....>>>>.....>>>>>>> L = "+L);
}
count += 2;
}
else if(currentPart == 3) //upper
{
    if(rest >= singleCrew.length)
    {
        L += rest;
        count++;
        System.out.println("7.....>>>>.....>>>>>>> L = "+L);
    }
    else
    {
        currentPart = 4;
        startY = 0;
        rest = deckLen;
        if(w < singleCrew.width)
        {
            rest -= (staircase.length + corridor);
            if(currentDeck < 2)
            {
                startY = grid[currentDeck].left;
                rest -= grid[currentDeck].left;
            }
            else
            {
                startY = (staircase.length + corridor);
                rest -= (staircase.length + corridor);
            }
        }
        subGridBreadth = 0;
    }
    L += deckLen;
    if(w < singleCrew.width) L -= 2*( staircase.length + corridor);
    System.out.println("8.....>>>>.....>>>>>>> L = "+L);
    count++;
}
else if(currentPart == 4) //lower
{

```

```

if(rest >= singleCrew.length)
{
    L += rest;
    count++;
    System.out.println("9.....>>>>.....>>>>>> L = "+L);
}
}
L -= ((count-1) * max(singleCrew.length, doubleCrew.length));
System.out.println("10.....>>>>.....>>>>>> L = "+L);
System.out.println("available length = " + L + " sl = " + singleCrew.length + " dl = " + doubleCrew.length +
"count = " + count);

int temp = L / noOfCrew;
System.out.println("temp = " + temp);

if(temp >= singleCrew.length)
{
    noOfSingleCrewModule = noOfCrew;
    noOfdoubleCrewModule = 0;

    officerModules[noOfRank] = singleCrew;
    noOfPersonInEachRank[noOfRank] = noOfSingleCrewModule;
    currentRank = noOfRank;
    noOfRank++;
    moduleNames[noOfRank] = "SM";
    unplaced = noOfSingleCrewModule;
    placed = 0;
}
else
{
    double ttmp = (L - (double)singleCrew.length*noOfCrew)/((double)doubleCrew.length - 2 *
singleCrew.length);
    noOfdoubleCrewModule = (int)Math.ceil(ttmp);
    noOfdoubleCrewModule = min(noOfCrew/2, noOfdoubleCrewModule);
    System.out.println("here....." + ttmp + " " + singleCrew.length + " " + noOfCrew + " " +
doubleCrew.length);
    if(noOfdoubleCrewModule >= 0)
    {
        noOfSingleCrewModule = noOfCrew - 2* noOfdoubleCrewModule;
        officerModules[noOfRank] = singleCrew;
        noOfPersonInEachRank[noOfRank++] = noOfSingleCrewModule;
        moduleNames[noOfRank] = "SM";
        officerModules[noOfRank] = doubleCrew;
        noOfPersonInEachRank[noOfRank++] = noOfdoubleCrewModule;
        moduleNames[noOfRank] = "DM";
        currentRank = noOfRank - 2;
        unplaced = noOfSingleCrewModule + noOfdoubleCrewModule;
        placed = 0;
    }
}

```

```

}
else
{
    flag = true;
    System.out.println("It is not possible to place the crew modules.");
    impossible = true;
}
}
System.out.println(".....s = " + noOfSingleCrewModule);
System.out.println(".....d = " + noOfdoubleCrewModule);
System.out.println(".....cd = " + currentDeck + " cp = " + currentPart + " cr = " +
currentRank + " rest = " + rest + " dl = " + deckLen);
System.out.println(".....up = " + unplaced);

if(noOfSingleCrewModule*singleCrew.length + noOfdoubleCrewModule*doubleCrew.length > L)
impossible = true;
if(!impossible)
{
    if(unplaced > 0)
    {
        placeCrewMod();
        currentDeck++;
        System.out.println("ki hoilo");
    }
    int tmw = (deckWidth - staircase.width - 2*corridor)/2;
    while( unplaced > 0 && currentDeck<noOfDeck)
    {
        startX = 0;
        if((currentDeck==0 || currentDeck==1) && tmw >= singleCrew.width)
        {
            currentPart = 1;
            startY = 0;
            rest = tmw;
            subGridBreadth = staircase.width;
        }
        else
        {
            currentPart = 3;
            if(tmw >= singleCrew.width)
            {
                startY = 0;
                rest = deckLen;
            }
            else
            {
                startY = staircase.length + corridor;
                rest = deckLen - 2*startY;
            }
        }
    }
}

```

```

        subGridBreadth = staircase.width;
    }
    placeCrewMod();
    currentDeck++;
}
printDecks(); //debug purpose
if(unplaced >0) flag = true;
//System.out.println("still unplaced " + unplaced);
}
else flag = true;
//System.out.println("It is not possible to place the modules in this template");
testCM();
for(int pq=0; pq<noOfDeck; pq++) updateCSumAll(pq);
printDecks();

int pn = noOfRank+1;
for(int i =0; i<noOfModules; i++)
{
    moduleNames[pn++] = allModules[i].sName;
}

//processCMFM();

unplaced = noOfModules;
System.out.println("nomod " + unplaced);
for(currentDeck=0; unplaced>0 && currentDeck<noOfDeck; currentDeck++)
{
    startX = grid[currentDeck].up;
    startY = grid[currentDeck].left;

    rest = deckLen - grid[currentDeck].right-grid[currentDeck].left;
    placeCMFM();
}
extra += unplaced;
if(unplaced > 0) flag = true;
drawDecks = new deckTabs();
if(flag)
    JOptionPane.showMessageDialog(null, "Placement is not possible for current template. Change the
template and try again.");
}
catch(Exception e){JOptionPane.showMessageDialog(null, "Error in processing.");}
//printDecks();
}
public static void testCM()
{
    int t = staircase.length + corridor;
    for(int i=0; i<noOfDeck; i++)
    {

```



```

if(grid[i].left <= t) grid[i].left = t;
if(grid[i].right <= t) grid[i].right = t;

if(grid[i].up <= corridor) grid[i].up = 0;
if(grid[i].down <= corridor) grid[i].down = 0;

startX = grid[i].up;
startY = grid[i].left;

System.out.println("l = " + grid[i].left + " r = " + grid[i].right + " u = " + grid[i].up + " d = " + grid[i].down);
//color(i, deckWidth - grid[i].down-grid[i].up, deckLen - grid[i].right-grid[i].left , 10);
}
}
public static void processCMFM()
{
    int i, j,k;

    for(i=0; i<noOfDeck; i++)
        for(j=0; j<noOfModules; j++)
            if(allModules[j].npDecks ==1 && allModules[j].pDeck[0] == i)
            {
                System.out.println("....." + i + " " + j);
                deckVsMod[i][counts[i]++] = j;
            }

    for(i=0; i<noOfDeck; i++)
        for(j=0; j<noOfModules; j++)
            if(allModules[j].npDecks > 1)
            {
                for(k=0; k<allModules[j].npDecks; k++)
                    if(allModules[j].pDeck[k] == i)
                        deckVsMod[i][counts[i]++] = j;
            }
}
public static void placeStairs()
{
    if(staircase.width % 2 == 1 )
    {
        staircase.width++;
        staircase.size = staircase.length * staircase.width;
    }
    int w1 = (deckWidth - staircase.width) / 2;
    int w2 = w1 + staircase.width;
    int x2 = deckLen - staircase.length;

    for(int k=0; k<noOfDeck; k++)

```

```

{
    grid[k] = new subGrids(0, 0, 0, 0);
    for(int i= w1; i<w2; i++)
        for(int j=0; j<staircase.length; j++)
            {
                decks[k][i][j] = 100;
                decks[k][i][x2+j] = 100;
            }
    }
}
public static void placeAM1()
{
    startX = 0;
    startY = 0;
    rest = (deckWidth - staircase.width - 2*corridor) / 2;
    int w1 = rest;
    currentPart = 1;
    while(unplaced > 0 && rest >=officerModules[currentRank].width)
        findModulePlaceX(rest);

    if(unplaced > 0)
    {
        startX = deckWidth - w1;
        startY = 0;
        rest = w1;
        currentPart = 2;
        while(unplaced > 0 && rest >=officerModules[currentRank].width)
        {
            System.out.println("w1 = " + w1 + " %%%%%here ");
            findModulePlaceX(rest);
        }
    }
    grid[currentDeck].left = subGridBreadth + corridor;
    grid[currentDeck].right = staircase.length + corridor;

    subGridBreadth = 0;

    startX = 0;
    startY = grid[currentDeck].left;
    int l1 = deckLen - grid[currentDeck].left;
    rest = l1;
    currentPart = 3;
    if(w1 < officerModules[currentRank].width ) rest -= (staircase.length + corridor);
    while(unplaced > 0 && rest >=officerModules[currentRank].length)
        findModulePlaceY(rest,0); // uppper portion

    grid[currentDeck].up= subGridBreadth + corridor;
    subGridBreadth = 0;

```

```

if(unplaced > 0)
{
    startY = grid[currentDeck].left; //startx variable
    rest = l1;
    currentPart = 4;
    if(w1 < officerModules[currentRank].width) rest -= (staircase.length + corridor);
    while(unplaced > 0 && rest >=officerModules[currentRank].length)
        findModulePlaceY(rest,1);
}
grid[currentDeck].down = subGridBreadth + corridor;
}
public static void placeAM2()
{
    grid[currentDeck].left = staircase.length + corridor;
    grid[currentDeck].right = staircase.length + corridor;

    System.out.println("===== " + grid[currentDeck].left);

    subGridBreadth = 0;

    startX = 0;
    startY = grid[currentDeck].left;
    int l1 = deckLen - grid[currentDeck].left - grid[currentDeck].right;
    rest = l1;
    int w1 = (deckWidth - staircase.width - 2*corridor) / 2;
    if(w1 >= officerModules[currentRank].width)
    {
        startY = 0;
        rest += 2*(staircase.length + corridor);
    }
    currentPart = 3;
    while(unplaced > 0 && rest >=officerModules[currentRank].length)
        findModulePlaceY(rest,0);

    grid[currentDeck].up= max(grid[currentDeck].up, subGridBreadth + corridor);
    subGridBreadth = 0;

    if(unplaced > 0)
    {
        startX = deckWidth - officerModules[currentRank].width;
        startY = grid[currentDeck].left;
        rest = l1;
        if(w1 >= officerModules[currentRank].width)
        {
            startY = 0;
            rest += 2*(staircase.length + corridor);
        }
    }
}

```



```

        rest -= (staircase.length+corridor);

case 3:
    while(unplaced > 0 && rest >=officerModules[currentRank].length)
        findModulePlaceY(rest,0); // upper portion

    if(currentDeck < noOfDeck )
        grid[currentDeck].up= max(grid[currentDeck].up, subGridBreadth + corridor);

    subGridBreadth = 0;
    if(currentDeck < noOfDeck && currentDeck < 2 && currentRank < noOfRank)
    {
        startY = grid[currentDeck].left;
        rest = deckLen - startY;

        if(w1 < officerModules[currentRank].width)
            rest -= (staircase.length + corridor);
    }
    else if(currentDeck < noOfDeck && currentRank < noOfRank)
    {
        startY = 0;
        rest = deckLen;

        if(w1 < officerModules[currentRank].width)
        {
            startY = staircase.length + corridor;
            rest -= 2*(staircase.length + corridor);
        }
    }
    if(currentRank < noOfRank)
        startX = deckWidth - officerModules[currentRank].width;

case 4:
    while(unplaced > 0 && rest >=officerModules[currentRank].length)
        findModulePlaceY(rest,1);
    if(currentDeck < noOfDeck )
        grid[currentDeck].down = max(grid[currentDeck].down, subGridBreadth + corridor);
        subGridBreadth = 0;
    }
}
public static void placeCMFM()
{
    System.out.println("in placeCMFM " + unplaced + " " + " "+currentDeck+ " " +counts[currentDeck]);
    int kk = 0;
    boolean yes = false;

    while(unplaced > 0 && kk<counts[currentDeck])
    {

```

```

    if(placedMod[deckVsMod[currentDeck][kk]])
    {
        kk++;
        continue;
    }
    tm = allModules[deckVsMod[currentDeck][kk]];
    minLen = 10000;
    yes = decideDim();
    if(yes)
    {
        int cl = deckVsMod[currentDeck][kk]+ noOfRank+1;
        placedMod[deckVsMod[currentDeck][kk]] = true;
        findCMFFMPlaceY(cl);
    }
    kk++;
}
}
public static boolean decideDim()
{
    System.out.println("in dec");
    int p, q, x, y;
    boolean tag = true;
    for(int i=0; i<tm.clw; i++)
    {
        p = startX;
        q = startY;

        x = tm.width[i] + corridor;
        if(startX+x > deckWidth+corridor) continue;
        else if(startX+x == deckWidth+corridor) x-= corridor;

        y = tm.len[i] +corridor;
        if(startY +y > deckLen-staircase.length) continue;

        System.out.println("in decision " + p + " " + q + " " + x + " " + y);
        if(isEmpty(currentDeck, p, q, x, y))
        {
            System.out.println(" yes");
            tag = false;
            if(tm.len[i] < minLen)
            {
                minLen = tm.len[i];
                modW = tm.width[i];
            }
        }
    }
}
if(tag) return false;
return true;

```

```

}
public static void color(int d, int x2, int y2, int c)
{
    try{
        System.out.println("*****in color = " + c + " deck = " + currentDeck + " startX
= "+ startX + " startY = " + startY + " x2 = " + x2 + " y2 = " + y2);
        for(int i=startX; i<startX + x2; i++)
            for(int j=startY; j<startY + y2; j++)
                {
                    if( decks[d][i][j] != 0)
                        flag = true;
                    decks[d][i][j] = c;
                }
            }
        catch(Exception e){ flag = true;
        }
    }
public static boolean isEmpty(int d, int p, int q, int x, int y)
{
    int i , j;
    //try{
    System.out.println("BBBBBBBBBBBBBBBB " + d);
    boolean t = true;
    for(i=p; i<p+x; i++)
        for(j=q; j<q+x; j++)
            {
                if(i>=deckWidth || j >=deckLen) return false;
                if(decks[d][i][j] != 0) {t = false; break; }
            }

    //}catch(Exception e) { // System.out.println(i + " " + j); }

    return t;
}

public static boolean isEmpty1(int d, int p, int q, int x, int y)
{
    int sum = c[d][x][y];
    if(p > 0) sum-= c[d][p-1][y];
    if(q > 0) sum-= c[d][x][q-1];
    if(p>0 && q>0) sum+= c[d][p - 1][q - 1];
    if (sum == 0) return true;
    return false;
}
public static void updateCSumAll(int d)
{
    for(int i=0; i<deckWidth; i++)c[d][i][0] = decks[d][i][0];
    for(int i=0; i<deckLen; i++)c[d][0][i] = decks[d][0][i];
}

```



```

    for(int i=1; i<deckWidth; i++)
        for(int j=1; j<deckLen; j++)
            c[d][i][j] = decks[d][i][j] + c[d][i-1][j] + c[d][i][j-1] - c[d][i-1][j-1];
}
public static void updateCSumPart(int d, int stx, int sty)
{
    if(stx == 0)
        for(int i=sty; i<deckWidth; i++)c[d][i][0] = decks[d][i][0];
    if(sty == 0)
        for(int i=stx; i<deckLen; i++)c[d][0][i] = decks[d][0][i];

    for(int i=stx; i<deckWidth; i++)
        for(int j=sty; j<deckLen; j++)
            if(i-1 >=0 && j-1 >= 0)
                c[d][i][j] = decks[d][i][j] + c[d][i-1][j] + c[d][i][j-1] - c[d][i-1][j-1];
}
private static void readExcelData(List sheetData) {

    String t1,t2;
    HSSFCell cell;
    List list;
    module tempMod;
    int i = 0, j, k, modSize;
    noOfModules = 0;

    //No. of deck
    list = (List) sheetData.get(i++);
    cell = (HSSFCell) list.get(1);
    noOfDeck = (int)Float.parseFloat(cell.toString().trim());
    System.out.println("noOfDeck = " + noOfDeck);

    //Deck area
    list = (List) sheetData.get(i++);
    cell = (HSSFCell) list.get(1);
    deckArea = (int)Float.parseFloat(cell.toString().trim());
    cell = (HSSFCell) list.get(2);
    deckLen = 2* (int)Float.parseFloat(cell.toString().trim());
    cell = (HSSFCell) list.get(3);
    deckWidth = 2*(int)Float.parseFloat(cell.toString().trim());
    System.out.println("deckArea = " + deckArea + deckLen + deckWidth);

    //No. of rank
    list = (List) sheetData.get(i++);
    cell = (HSSFCell) list.get(1);
    noOfRank = (int)Float.parseFloat(cell.toString().trim());
    System.out.println("noOfRank = " + noOfRank);
}

```

```

//No. of personnel in each rank (1 -> 2 -> ...)
noOfOfficerModules = 0;
moduleNames[0] = "ST";

list = (List) sheetData.get(i++);
for (j = 1; j < list.size(); j++)
{
    cell = (HSSFCell) list.get(j);
    noOfPersonInEachRank[j-1] = (int)Float.parseFloat(cell.toString().trim());
    moduleNames[j] = "AM Rank " + j;
    System.out.print(" np = " + noOfPersonInEachRank[j-1]);
    noOfOfficerModules += noOfPersonInEachRank[j-1];
}
System.out.println("noofficermodule = " + noOfOfficerModules);

//No. of crew
list = (List) sheetData.get(i++);
cell = (HSSFCell) list.get(1);
noOfCrew = (int)Float.parseFloat(cell.toString().trim());
System.out.println("noOfCrew = " + noOfCrew);

//Single crew module size
list = (List) sheetData.get(i++);
cell = (HSSFCell) list.get(1);
singleCrewModuleSize = (int)Float.parseFloat(cell.toString().trim());
System.out.println("singleCrewModuleSize = " + singleCrewModuleSize);
singleCrew = new fixedModule(singleCrewModuleSize);

//Double crew module size
list = (List) sheetData.get(i++);
cell = (HSSFCell) list.get(1);
doubleCrewModuleSize = (int)Float.parseFloat(cell.toString());
System.out.println("doubleCrewModuleSize = " + doubleCrewModuleSize);
doubleCrew = new fixedModule(doubleCrewModuleSize);

//Officer module size
list = (List) sheetData.get(i++);
for (j = 1; j < list.size(); j++)
{
    cell = (HSSFCell) list.get(j);
    officerModuleSizes[j-1] = (int)Float.parseFloat(cell.toString().trim());
    System.out.print(" om = " + officerModuleSizes[j-1]);
    officerModules[j-1] = new fixedModule(officerModuleSizes[j-1]);
}
System.out.println("");

//Fixed module specification

```



```

    modSize = (int)Float.parseFloat(cell.toString().trim());
    System.out.println("module name = " + t1 + " deck = " + t2 + " ModSizes = " + modSize);
    if(i<=14)tempMod = new module(modSize); //fm
    else tempMod = new module(modSize); //cm
    tempMod.moduleName = t1;
    tempMod.sName = un[i-11];

    if(t2.equals(""))
    {
        tempMod.npDecks = noOfDeck;
        for(int ts = 0; ts<noOfDeck; ts++){
            tempMod.pDeck[ts] = ts; // ts = deck no
            deckVsMod[ts][counts[ts]++] = noOfModules;

            System.out.print(ts + " ");
        }
    }
    else
    {
        tempMod.npDecks = 1;
        int nn = t2.charAt(0) - 'A';
        int nnd = noOfDeck - nn - 1;
        tempMod.pDeck[0] = nnd;
        System.out.print(t2.charAt(0) + " nn " + nn + " " + nnd);
        deckVsMod[nnd][counts[nnd]++] = noOfModules;
    }
    allModules[noOfModules++] = tempMod;
}System.out.println("");
}
System.out.println(noOfModules);
}

public static int min(int a, int b)
{
    if (b < a) return b;
    return a;
}
public static int max(int a, int b)
{
    if (b > a) return b;
    return a;
}
public static void printDecks()
{
    for(int i=0; i<noOfDeck; i++)
    {
        for(int j=0; j<deckWidth; j++)
        {
            for(int k = 0; k<deckLen; k++)

```

```

        System.out.print(decks[i][j][k]);
        System.out.println();
    }
    System.out.println(); System.out.println();
}
}
}

```

Module class

```

package dssupd;
/*@author Choudhury */

public class module {

    String moduleName = new String();
    String sName = new String();
    int size;
    int len[] = new int[50];
    int width[] = new int[50];
    int mindim;
    int pDeck[] = new int[20]; // possible decks where the module can be placed
    int npDecks;
    int clw;

    module(){}
    module(int s)
    {
        size = s * 4;
        mindim = (int)(Math.sqrt(size) * 9 / 13);
        int sq = (int)(Math.sqrt(size));
        clw = npDecks = 0;

        for (int i = mindim; i <= sq; i++)
        {
            if (size % i == 0 && (size/i) >= mindim)
            {
                len[clw] = i;
                width[clw] = size / i;

                if (len[clw] != width[clw])
                {
                    clw++;
                    len[clw] = size / i;
                    width[clw++] = i;
                }
                else clw++;
            }
        }
    }
}

```

```

if(clw == 0)
{
    int l = (int)Math.floor(Math.sqrt(size));
    int u = (int)Math.ceil(Math.sqrt(size));

    int ll = size - l * l;
    int uu = u * u - size;

    if (ll < uu )
    {
        len[clw] = l;
        width[clw] = l;
        size = l*l;
    }
    else
    {
        len[clw] = u;
        width[clw] = u;
        size = u*u;
    }
    clw++;
}
for (int j = 0; j< clw; j++)
    System.out.println("len = " + len[j] + " wid = " + width[j]);
}
}

```